Data-Driven Supervision of Autonomous Systems

Davide Dell'Anna

Utrecht University

SIKS Dissertation Series No. 2021-05
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover by Lorenzo Dolfi
Printed by Ridderprint | www.ridderprint.nl

ISBN 978-94-6416-395-7

Data-Driven Supervision of Autonomous Systems

Datagestuurde Supervisie van Autonome Systemen

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit Utrecht
op gezag van de
rector magnificus, prof.dr. H.R.B.M. Kummeling,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen op
woensdag 10 maart 2021 des middags te 2.30 uur

door

DAVIDE DELL'ANNA

geboren op 29 augustus 1991
te CHIVASSO, Italië

Promotoren:	Prof. dr. M.M. Dastani
	Prof. dr. S. Brinkkemper
	Prof. dr. J.-J.Ch. Meyer
Copromotor:	Dr. F. Dalpiaz

Assessment Committee:

Prof. dr. Michael Luck, King's College London
Dr. Anna Perini, Fondazione Bruno Kessler
Prof. dr. Amal El Fallah Seghrouchni, Sorbonne University
Prof. dr. Munindar P. Singh, North Carolina State University
Prof. dr. Leon van der Torre, University of Luxembourg

Contents

1 Introduction

Software is an integral part of our daily lives. Not anymore confined to computational settings where it works in isolation, software is now often part of broader systems, where continuous interaction between a multitude of interconnected entities takes place. In a modern city, for example, software is embedded in speed cameras, traffic lights, electronic signs and autonomous vehicles, and enables them to interact not only with the surrounding environment, but also with each other, and with us. In a smart home, cameras, light and temperature sensors, smart TVs and speakers, continuously interact with human actors, with their operating environment, and with each other.

As systems become more interconnected and diverse, engineers are less able to anticipate and model all possible interactions that will take place among components. The term *unknowns* is sometimes used to refer to the boundaries of knowledge that is available during system design [263]. Different types of unknowns include the *known unknowns*, which concern properties that cannot be assessed during the system design, but will need to be monitored during execution, or the *unknown knowns*, which refer to tacit knowledge. The *unknown unknowns*, finally, concern situations that cannot be even predicted to happen at some point in the future, so they are not considered during the system design. Unknowns are common for modern software systems, which operate in increasingly dynamic settings [258].

When unforeseen situations occur, *assumptions* made during the design of the system may become invalid in the new operating *context*, thereby causing the system to fail satisfying its *requirements* and meeting the system's *objectives* [56].

Despite their dynamism and complexity, software systems are still expected to perform optimally and to continuously maintain their fitness within a changing environment [191]. In order to deal with changes, however, it is often unrealistic to consider maintenance in a classical fashion, where, for example, the system is halted and software engineers intervene to restore its compliance with the requirements [34, 196]. This is especially true for systems that continuously grow in size and complexity and for which timely and decisive responses are expected.

In recent years, the concept of *autonomic computing* has gained growing attention as a paradigm for coping with the high run-time uncertainty and unpredictability of modern software systems [175]. The term refers to computing systems that can manage themselves, given a specification of the system's objectives. Autonomic com-

puting systems are endowed with self-configuring, self-optimizing, self-healing and self-protecting *run-time* capabilities, that allow them to autonomously re-configure their components and sub-systems; to continuously attempt to improve their performance and efficiency; to automatically perform monitoring, diagnosis and reparation of their components; and to try to anticipate and mitigate possible problems. These capabilities are meant to guarantee that the software can autonomously adapt to the evolving circumstances, without the need of continuous human intervention.

Several solutions have been proposed in the literature to provide software systems with such capabilities. Most of them focus on self-adaptation aimed at restoring the compliance of the system with its requirements [32, 113].

In highly dynamic contexts, however, the system's requirements may themselves result inadequate at run-time to guarantee the overall system objectives [15, 192]. For example, the enforcement of *predefined* speed limits on vehicles in a smart city may not guarantee adequate traffic throughput or safety if the weather conditions change, or if an increasing number of autonomous vehicles are introduced on the roads. Moreover, since modern software systems are not isolated from the rest of our society, they are also subject to the dynamics and continuous evolution of our society [238]. The introduction of new government regulations about CO_2 emissions to cope with the global warming problem, for example, may affect the adequacy of previously defined traffic regulations. The objectives of the system themselves therefore can change, for new needs arise, others are dropped, the desired quality requirements vary, and the relative priorities change over time [134, 309]. When this happens, previously defined requirements may become inadequate to guarantee the achievement of the new system's objectives. Capabilities of *adaptation at the requirements level* are paramount in modern software systems [133]. Such capabilities define the broad scope of this thesis.

One of the distinguishing features of modern software systems is the complex interplay that takes place between the technical components and the human (social) actors. Systems such as a smart city or a smart home are often referred to as *Socio-Technical Systems* (STSs) [36]. STSs are heterogeneous systems, where the participants are *autonomous* and linked via *social* dependencies, rather than through hardwired connections. In STSs, the participants, think for example of self-driving cars in a smart city, are themselves *autonomous systems*, i.e., systems that act independently of continuous and direct human control, and they are part of a larger system which they may enter or leave at any time. This is one of the factors that contributes to make the operational environment volatile and highly dynamic and makes it highly complex, if not impossible, to anticipate and model all the possible states of the system [100]. The autonomy of the involved entities, furthermore, makes them *weakly controllable* [81], so that it is not always possible to anticipate their behaviors and to guarantee their compliance with the system's requirements [190, 294].

In designing these systems, not only the technical components need to be considered, but requirements shall also aim at *regulating the behavior* and interactions of the social actors with the technical ones, and with each other. Such requirements need to be an *explicit* part of the system, so that the involved actors can be aware of them and react to their changes. Their satisfaction does not depend solely on a given implementation, but also on the reaction to them from the social actors that operate

in the system, and on their willingness to comply with them.

Inspired by our society, where *norms* play an essential role in coordinating the behavior of individuals [130, 255], *Multi-Agent Systems* (MASs), and in particular *Normative MASs* (NMASs), have been proposed as a computational abstraction for dealing with (social) complex systems [279, 284] and STSs [256]. In a MAS, *agents* are autonomous heterogeneous entities, typically assumed to act according to their own preferences or goals, which reflect the goals of the users or owners they operate for in the system [298]. Agents' internals are typically unknown to other agents and to the designer of the overall system. In order to guarantee that the objectives of the system are achieved, norms and *sanctions* are typically used as a means to coordinate and regulate agents' behavior without over constraining their autonomy. From a system's perspective, norms in a MAS are enforced by a so-called *institution*, an exogenous (to the agents) entity with the task of monitoring and enforcing norms to regulate the behavior of the agents in order to guarantee, or improve, the achievement of the system's objectives.

In our attempt to develop adaptation mechanisms at the requirements level, in this thesis we also consider NMASs as a computational model for complex modern systems such as STSs. We focus on requirements aimed at regulating the behavior of the autonomous agents in the system. In this dissertation, we identify such type of requirements with norms, and we use the two terms interchangeably, unless specified otherwise. We study how to endow a NMAS with a *supervision* mechanism that allows an *institution* to autonomously revise, when necessary, the norms that are being enforced on the autonomous agents. For example, we consider a NMAS such as a smart road, where traffic rules (norms) regulate the behavior of self-driving cars (autonomous systems). The supervision of the smart road consists of monitoring the behavior of the self-driving cars in the system and automatically revising the enforced traffic rules, when necessary.

1.1 Motivation and Research Objectives

When designing a software system, the *requirements*–i.e., what the system and its components must do and what qualities they must have–are identified based on the objectives of the different stakeholders (we call them *system's objectives*) and on *assumptions* which depend on the available domain knowledge. The requirements are then reflected in a *specification*, which describes design decisions on how to satisfy the requirements. Using Zave and Jackson's traditional formulation of the requirements problem [306], which they indicate as $W, S \vdash R$, the specification S, given some assumptions W, must be sufficient to satisfy the requirements R.

Over the years a wide number of model checking and verification techniques have been developed to formally verify and prove the correctness of a defined system's specification with respect to its requirements [35, 228] and to satisfy properties such as liveness or safety properties [9, 109, 178].

Traditional software, confined to relatively static and well-defined domains, allows to model all aspects of the system and to perform formal *verification* of its specification with respect to the requirements. When dealing with the highly dynamic and ever-

evolving software systems that are embedded in our modern society, however, formally proving that a specification satisfies the requirements, does not suffice anymore, on its own, to guarantee that a system achieves its objectives (referred to as software validation [60]).

In his seminal work [189], Lehman illustrates that for modern evolving software systems, which he calls *E-programs* and classifies as part of a broader class of dynamic programs called *A-programs*, the correctness of the implementation is only partially relevant, because the *validity* and *effectiveness* of the system is not assessed anymore solely with respect to its specification, but depends also on human satisfaction about the *overall behavior* of the software. Quoting Lehman,

> "Correctness and proof of correctness of the program as a whole are, in general, irrelevant in that a program may be formally correct but useless, or incorrect in that it does not satisfy some stated specification, yet quite usable, even satisfactory. Formal techniques of representation and proof have a place in the universe of A-programs but their role changes. It is the detailed behavior of the program under operational conditions that is of concern."

Furthermore, the complexity and volatility of modern software systems and of the environment in which they operate make it often too hard—or practically impossible—to predict a priori (i.e., at *design-time*) the effectiveness and validity of specific requirements in all (possibly evolving) environmental conditions [15, 192]. System designers inevitably have to rely on a number of assumptions about what is expected from the operating environment, about the possible interactions that can happen between different software components, or about the conditions when certain requirements are satisfied. When the context changes at *run-time*, the design-time assumptions may become invalid. Invalid design-time assumptions, have been identified among the main risk factors for error and loss in software development [55].

The literature on self-adaptive systems offers good solutions to adapt a system when their requirements are threatened [41, 113, 185]. Generally speaking, the literature so far mainly focused on how to respond to changes in the environment by re-configuring the components and the architecture of the system, or, for security systems, by halting the system in case of noncompliance with the requirements [34, 196]. In a modern software system like a STS, however, control over all system's components is often impossible. The involved components, for example, may be services which autonomously run in remote and belong to different stakeholders [33]. As a result, if the requirements of the system are not met at run-time, altering its components to restore the compliance with the requirements is not always an option [81]. Additionally, since the objectives of the stakeholders of the system may change over time [134, 309], the previously defined requirements may become ineffective to guarantee the new objectives [15].

In these situations, run-time revision of requirements is one of the key factors to build a versatile system capable of ensuring the stakeholders objectives [180]. Temporarily *relaxing* a strict requirement in previously unpredicted operating conditions, and *learning* under which conditions the requirements are more useful, may guarantee the stability of the system without the need of an adaptation of its components. Revis-

ing the requirements is justified if guided by the stakeholders objectives [21, 275, 303]. For example, in an urban transportation software system, the city council may have the objective that *every day, at most x car accidents should occur, possibly none*, and a requirement for the drivers may be that *vehicles shall possess an automatic braking system*. But one may as well replace this requirement with *vehicles shall possess a cruise control system*.

Motivated by early studies on requirements relaxation and approximation [10, 294], we propose a framework for the *supervision* of software systems. Our framework aims at providing a system with requirements level supervision capabilities, i.e., with the capability of *automatically revising its requirements* (norms in NMAS terminology) when, due to the evolution of its operating context, there is *evidence* that the current requirements are not effective anymore in guaranteeing the system's objectives. In particular, we consider situations where a model of the system is not available, and the evidence about the effectiveness of the requirements can be obtained only at run-time and through learning from *execution data* [176]. The main focus and contributions of this thesis, therefore, concern a *data-driven* run-time supervision framework that continuously monitors the execution of software systems, evaluates their behavior against the requirements and the system's objectives, and intervenes by deciding which requirements should be revised and how.

The main research objective that we aim to achieve is the following.

> **Research Objective.** *To design a data-driven run-time requirements revision framework that ensures a sufficient achievement of the objectives of software systems that operate in dynamic, evolving and weakly controllable environments.*

In the framework that we envision, which characterizes the main components and features required for providing a system with run-time requirements revision capabilities, a revision of the current requirements is identified at run-time whenever the system's objectives are not sufficiently met. The first step is to evaluate the design-time assumptions to diagnose the current requirements and to identify the root causes of the problem. Based on the results of the diagnosis, the framework should determine how the requirements should be revised. For example, whether a requirement should be relaxed, made more strict, or completely changed. In other words, the framework should determine the *direction*, or *type*, of the revision of the requirements. In a data-driven supervision framework that also intends to be transparent and explainable, the type of revision should be identified according to some given data-driven policy or strategy. For example, a simple high-level data-driven policy could be to *relax a requirement R if, according to the data, the system's objectives are more likely to be achieved when R is violated*. The current requirements should then be revised in line with the policy. *Revision operations* should be performed on the current requirements to obtain new requirements in line with the policy. The choice of the new requirements determines the *intensity* of the revision. If more than one option is available for the new requirements, the selection should consider different criteria. These criteria should include, for example, the expected improvement of the system objectives' achievement, or the similarity with the current requirements (choosing very

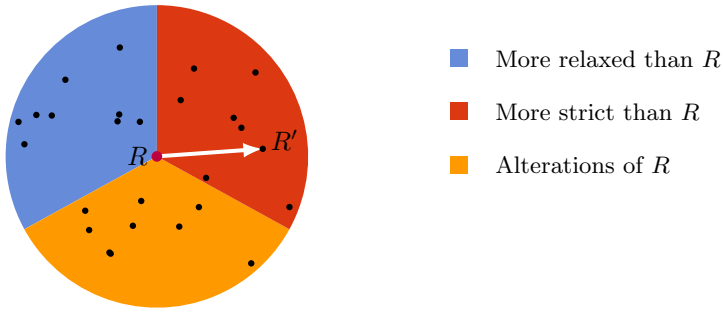


Figure 1.1: *A simplified illustration of a discrete space of possible revisions of a requirement R. Each black dot is a possible revision of R, and belongs to one of three categories: More relaxed than R, More strict than R, and Alterations of R. The arrow indicates the chosen revision R′; its direction denotes the type of revision (More strict than R, in the figure); its length denotes the intensity of the revision (the closest to R the lowest the intensity) according to some given metric.*

different requirements from the current ones may affect, for example, the stability of the system). Fig. 1.1 provides a simplified illustration of the concepts described above.

In order to achieve our research objective, we identify four research questions.

RQ 1. *How to validate at run-time the design-time assumptions that are reflected in the requirements?*

In designing a software system, engineers are forced to, explicitly or implicitly, make assumptions that are then reflected in the system's requirements and specification [191]. Invalid design-time assumptions have been identified as one the main causes for software evolution [15]. This is particularly true for complex modern systems, where the autonomy of the participating entities and the dynamic and ever-evolving operating environment lead to heightened run-time uncertainty [16, 294]. Assessing at run-time the validity of the design-time assumptions is crucial to diagnose the requirements and to establish the need for their revision [15]. We study how to validate with run-time data a variety of design-time assumptions that are reflected in the defined requirements.

RQ 2. *Which are possible data-driven requirements revision policies?*

Requirements revision policies should indicate which type of revision of the requirements is needed, and in which cases. In our framework, we are interested in data-driven policies, i.e., policies that make use of data about the run-time satisfaction of the requirements and the achievement of the system's objectives in the different operating contexts. Based on the acquired execution data, the policies should determine whether the requirements should be relaxed, made more strict, or if they require a more general alteration (one of the three areas in Fig. 1.1). Bayesian Networks [240]

have been widely used for performing diagnosis and automated decision making in a transparent and explainable way in different fields, ranging from medicine to forensic to software engineering [141, 147]. In this thesis, we make use of such graphical probabilistic model to collect and reason about run-time data, and we study a number of general and domain-independent policies that are based on probabilistic diagnosis and inference.

RQ 3. *Which are possible operations to revise requirements at run-time?*

The envisioned framework should be able to automatically perform operations of revision of the system requirements when necessary, i.e., it should be able to determine a new requirement in the chosen area in Fig. 1.1. As we will detail in Sec. 2.3, in the requirements engineering literature, *requirements models*, have been widely employed to represent the requirements of a system and to organize them in hierarchical structures [271]. We study possible run-time operations for revising a requirements model at run-time into more relaxed, more strict or simply different variations of the original one. We also consider the revision of specific requirements besides them being part of a requirements model. In doing so, we use *norms* and the normative concepts of *sanctions* and *rewards* to define requirements for the behavior of autonomous agents. Norms with sanctions permit to achieve a weak notion of control [50] of the weakly controllable (social) agents that are involved in complex modern systems [36, 258]. In the NMAS research literature, *conditional norms*, which include conditional prohibitions and conditional obligations, are commonly used to regulate the behavior of the autonomous agents [10]. We study which are possible run-time operations for revising the components of conditional norms, i.e., the condition of applicability of the norm, the main content of the norm, and its deadline, as well as how to determine opportune sanctions at run-time.

RQ 4. *How well does the proposed run-time requirements revision framework perform?*

To evaluate our framework we shall assess its effectiveness in determining at run-time new requirements that ensure the achievement of the system's objectives. We will evaluate our framework, and the proposed data-driven requirements revision policies and revision operations, in terms of a variety of criteria. These will include the degree of *achievement of the system's objectives* with the identified requirements; the *speed* in identifying requirements that ensure a sufficient achievement of the objectives; the *stability* of the system when subject to the revisions; the *alignment* of the requirements with the system's objectives, and the *resilience to noise* in the monitored data. We will conduct extensive experimental evaluations of our framework by means of *simulation*, a powerful tool for evaluating prototypical novel solutions [149]. In particular we will consider traffic simulation, since the state of the art solutions, such as the SUMO open source simulator [184], provide realistic settings for smart road scenarios, which are relevant for the scope of this thesis.

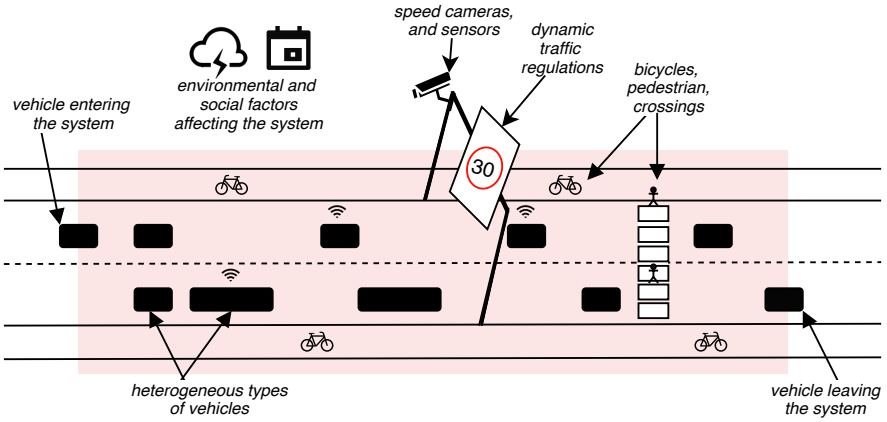


Figure 1.2: *A sketch of the multitude of entities involved in a smart road. The red area indicates the boundaries of the system being supervised. Vehicles are represented as black boxes to emphasize that their internals are not known.*

1.2 Smart Road Scenarios

The solutions proposed in this thesis are meant to be general and applicable to different types of systems. As we will see, the approach and strategies that we propose are domain independent and do not rely on domain-specific knowledge. Throughout this thesis, however, we are going to focus on the particular application area of *smart roads*. We will make use of different illustrative case studies in such application area as plausibility probe studies [193] for testing and refining our proposal. The reason why we focus on this particular application area is threefold, as explained below.

Representativeness. Smart roads are an application scenario that exhibits the main features of modern complex systems such as the STSs described in the introduction of this thesis. Smart roads are complex systems where a multitude of autonomous and heterogeneous entities continuously interact and operate in a shared environment that is subject to continuous evolution. Fig. 1.2 provides an illustration of a simplified smart road. Vehicles can be seen as the autonomous and weakly controllable (software) agents, whose behavior we desire to regulate in order to achieve system's objectives such as desired road throughput, CO_2 emissions, or safety levels. Each of the agents can enter or leave the system at any moment, and it is characterized by its own preferences, goals, and internals, which are not necessary aligned with, nor known to, the other agents and the designer of the system (e.g., the city council). The environment in which these agents operate involves not only the vehicles, but also other technical and social entities such as pedestrian, bicycles, traffic lights, speed cameras. Furthermore, the system is affected by natural events, such as storms, snow, etc., as well as by social events, such as a football match, that affect the behavior of all the involved agents. Adequately modeling such a complex system is not a trivial task and requires many assumptions. The designed traffic regulations are therefore necessarily prone to errors and need to be continuously reassessed as the operating

context evolves. Consider, for example, the introduction of new regulations about CO_2 emissions in order to cope with the global warming problem. Previously defined speed limitations, may not be in line with the new system's objectives, and may need to be changed. In March 2020, for example, the Dutch government reduced the motorway speed limit from 120 km/h to 100 km/h as part of a package of measures aimed at reducing nitrogen precipitation in nature [235]. Traffic regulations need to be continuously evaluated at run-time, and possibly revised when they are not effective in achieving the desired system's objectives.

Relevance. As of today, the traffic problem is still an intolerable burden in many contexts [219]. Current traffic control systems enforce traffic rules, such as speed limitations, to regulate the behavior of drivers in order to achieve high throughput and to maximize safety. In the near future, an increasing number of autonomous vehicles (e.g., smart cars) are expected to populate our roads [2]. It is however unlikely that the companies that produce such vehicles will be sharing or agree on a common source code. Similarly to human drivers, no complete control will be possible over them, and neither a reliable prediction of their behavior nor their compliance with traffic rules could always be guaranteed. Differently from human drivers, however, autonomous vehicles may be able to handle continuous and dynamic information regarding the traffic regulations taking place. Smart roads represent a proposed infrastructural solution to provide autonomous vehicles with continuous information and directives so to achieve desirable system-level properties [280]. A smart road infrastructure, having information about the global situation, can issue customised directives to the autonomous vehicles, e.g., prescribing vehicles in a particular road to slow down, or vehicles in another road to accelerate, in order to optimise traffic across the whole network. Such directives, which may be enforced by means of sanctions on the autonomous vehicles to guarantee a desired degree of obedience, shall be continuously adapted to the evolving operating contexts, making *smart roads infrastructures* an ideal application for the run-time revision mechanisms studied in this dissertation.

Evaluation. Any technical contribution needs evidence in order to be evaluated. In our case, we aim at designing and developing solutions to change the requirements of a system into new ones at run-time and based on data. We need to be able to evaluate in some way the strategies that we propose to revise the requirements and the new revised requirements. The traffic domain gives us the possibility to do so by means of simulation. A number of traffic simulators have been developed in the years both for research and industrial purposes and provide realistic vehicle's models and complex dynamics. The SUMO Traffic Simulator [184], for example, is one of them. SUMO is an open source traffic simulator that is designed to handle large networks, and it has been used in numerous real-world and research applications [46, 85, 86, 184]. In the Software Engineering for Adaptive and Self-Managing Systems (SEAMS) research community, SUMO has been used as a basis of an exemplar specifically proposed as a case study for evaluating different self-adaptation techniques [245]. Moreover, SUMO is a microscopic traffic simulator. A microscopic simulator permits to model and consider the detailed behavior of each individual elements of a transportation system, for example allowing to model norm-aware agents that autonomously react to the en-

forcement of different norms and requirements. Microscopic approaches to simulation are opposed to macroscopic ones, which are concerned with the general distribution of vehicles, the density of traffic, and other macroscopic aspects of the system that abstract from the detailed behavior of individual agents. In this dissertation, we will make extensive use of traffic simulation in order to evaluate our proposals. Using SUMO allows us to validate our hypotheses with microscopic simulations that are realistic for smart roads scenarios involving autonomous vehicles.

1.3 Approach Overview and Contributions

The research method that we followed for this dissertation is based on the design science research methodology [295]. The first step was the *investigation* of the most common and relevant scenarios. We made use of the major case study of smart road and traffic regulation for smart cities, where different autonomous vehicles coexist in a shared environment and interact to perform their own tasks, and where revision of the requirements is necessary to achieve system-level objectives, such as maximizing road throughput, ensuring safety of drivers and pedestrians, or minimizing CO_2 emissions. After surveying the existing applicable state-of-art solutions in the literature, the major existing problems and limitations were identified for the *problem context*, and the *research questions* were defined. An *artifact*, the run-time data-driven supervision framework, was therefore designed in order to overcome the existing gap, by studying the relationship between the proposed solution and the context of the problem. Both theoretical analysis and the described case studies were used to validate the designed artifact and to trigger changes in areas that required improvements, guided by the research questions. Finally, the framework was *evaluated* throughout the whole process with the help of simulation.

We briefly outline the building blocks of our framework, thereby providing a concise overview of how it contributes to the state of the art.

Fig. 1.3 provides an high-level architectural overview of our framework. In line with the autonomic computing and self-adaptive systems literature [142], we make use of a general architecture for system run-time self-adaptation described as a *closed-loop control system*. In a control loop, data from sensors monitoring the system enters a controller, which continuously compares the received data with some desired values which reflect the system's objectives. The controller generates then an output that is aimed at reducing the difference between the monitored and the desired values.

In our framework, at design-time, the system *designers* define *requirements* based on the *objectives* and values of the *stakeholders* of the system and on *domain assumptions*. For instance, in a smart road scenario, the objectives of the city council, the stakeholders of the smart road, may include to maximize throughput and safety while minimizing CO_2 emissions. Based on such objectives, requirements concerning for example the speed of the vehicles on the road, their safety distance or the type of vehicles allowed in the road, are designed.

The system, indicated in Fig. 1.3 by the red area, is built according to the defined requirements and it is instrumented for their *Monitoring and Enforcement*. For

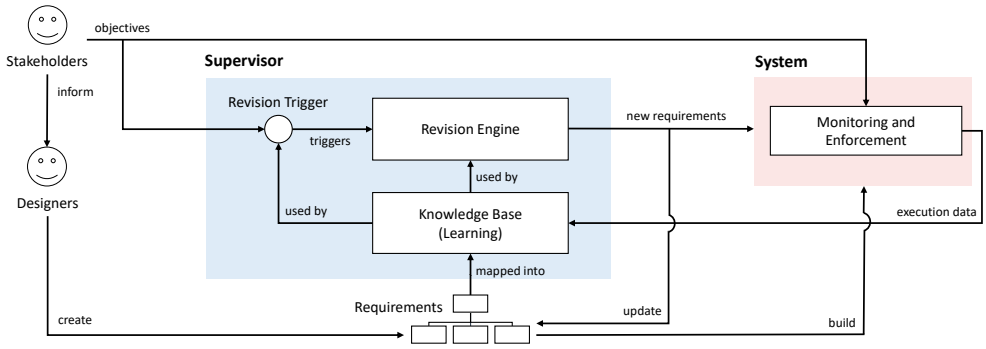


Figure 1.3: *An high-level architectural overview of our framework for the run-time data-driven supervision of autonomous systems. The blue area indicates the Supervisor component which is the main focus of this thesis. The red area indicates the system being supervised, for example the smart road in Fig. 1.2.*

instance, in the smart road scenario, the smart road infrastructure is equipped with smart cameras, traffic lights, dynamic information panels, traffic enforcement cameras, etc., as illustrated in Fig. 1.2.

At run-time, the *Monitoring* component of the system collects *execution data* about three main elements: (i) the satisfaction or violation of the requirements, (ii) the operating contexts in which they are evaluated (e.g., the hour of the day, or the weather conditions), (iii) the achievement of the system's objectives. Notice that we make a key distinction between the requirements and the objectives of the system, as we associate the latter to the *raison d'être* for the requirements [303]. Objectives of the systems are not always computationally verifiable (e.g., if they require to assess user satisfaction), and their evaluation may be obtained from the stakeholders in a discontinuous way. However, we assume they are measurable. Data collected by the monitoring component during system execution is stored into a *knowledge base* and used for *learning* the correlation between the elements (i), (ii) and (iii) described above. The *Revision Engine* component makes use of the information learnt in order to evaluate the validity of the design-time assumptions, and to decide if and how to revise the current requirements. A revision of the requirements is *triggered* when the current requirements do not guarantee a sufficient achievement of the objectives of the system. The supervision of the system consists of automatically revising the requirements in order to re-align them with the system's objectives, when deemed necessary.

The contributions of this thesis to the state of the art concern the *Supervisor* component in Fig. 1.3, and they can be summarized as follows. Fig. 1.4 relates the contributions to the main elements of the supervisor and to the chapters of the thesis.

- *Data-driven assumptions validation* [RQ 1, Chapter 3]. We propose a run-time approach to validate the assumptions made at design-time when engineering the system's requirements. In particular, we propose a mechanism that maps automatically a requirements model to a probabilistic graphical model which reflects the structure of the requirements model. The probabilistic graphical

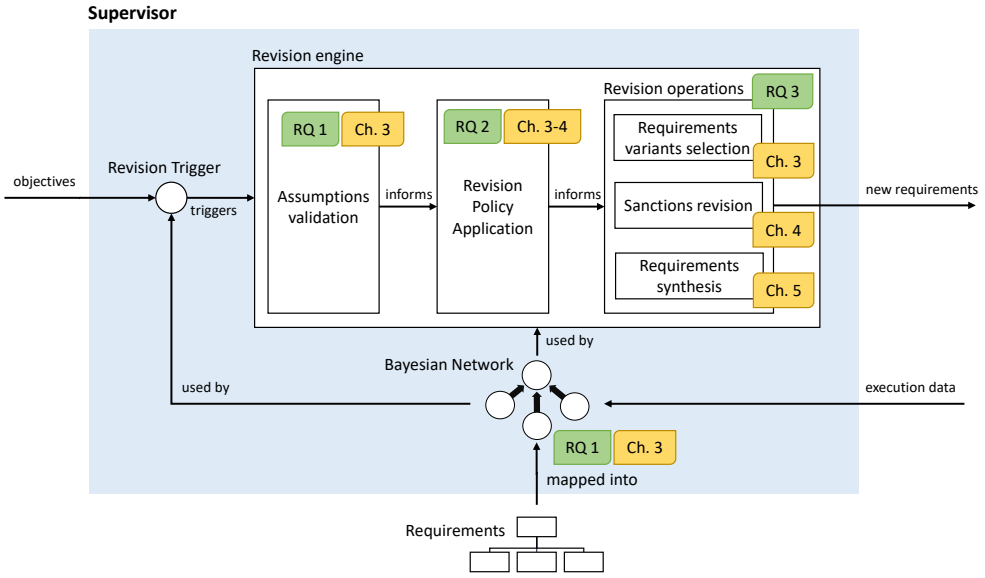


Figure 1.4: *An overview of the main contributions of this thesis.*

model (specifically, a Bayesian Network [240]) is used then at run-time to collect data and to learn statistical relationships between the requirements and the system's objectives in different operating contexts, as per the knowledge base in Fig. 1.3. We use the learned knowledge to provide a quantitative estimation of the degree of validity of the design-time assumptions of a requirements model, based on the acquired data. For example, in order to estimate the validity of the assumption that the satisfaction of a requirement R positively contributes to the achievement of a system's objective O, we compare the learned probability that O is achieved when R is satisfied, with the learned probability that O is achieved when R is violated.

While early work characterized the design-time assumptions of a requirements model [16], no practical run-time approach has been proposed to provide their probabilistic evaluation. Bayesian Networks (BNs) have been widely employed in Requirements Engineering for a variety of tasks [115], including the run-time verification of requirements [141]. Wu *et al.* [299] propose a preliminary study of the relationship between an iStar [101] requirements model and BNs in the context of requirements elicitation. Inspired by such study, we propose a formal and fully automated mapping, and an expressive type of BN which integrates contextual information and the possibility of disabling nodes, enabling its run-time use to validate design-time assumptions based on data.

- *Requirements revision policies* [RQ 2, Chapters 3-4]. We propose a variety of novel policies (which sometimes we will call *strategies*) that, by analyzing the statistical data acquired at run-time, diagnose the requirements and suggest how to revise them.

Most existing approaches to self-adaptation focus on reconfiguration of a system so to restore its compliance with requirements, assuming their correctness and effectiveness in achieving system's objectives. Existing work concerning requirements revision typically reassess the weights of non-functional requirements [17, 41]. We focus instead on strategies for the revision of requirements (or norms) so to re-align them with the system's objectives. We maintain at run-time an explicit representation of the system's objectives, characterizing the rationale behind the enforced requirements. Requirements and norm revision are driven by the system's objectives and they are based on the statistical knowledge learned at run-time from system execution data. In our framework, we do not assume correctness and effectiveness of requirements. In fact, data could show us that certain requirements are noneffective, or even harmful, in certain operating contexts for the achievement of the system's objectives, and need to be revised or ignored.

- *Requirements revision operations* [RQ 3, Chapters 3-5]. We propose algorithms to revise requirements based on run-time execution data. In particular, we propose novel algorithms to automatically select or synthesise more relaxed, more strict, or simply different variants of both requirements models and conditional norms with deadlines and sanctions.

  In the context of norm revision, we propose a novel approach to identify sanctions that effectively motivate an adequate portion of the population of agents to comply with the norms. In our approach, we make use of the relationship between the preferences of rational agents, their run-time behavior, and the run-time achievement of the system's objectives. Furthermore, while early work on the revision of conditional norms was presented in the literature in the context of monitor synthesis [10], no concrete algorithms were available to revise such norms in order to improve the norm alignment with system's objectives based on run-time execution data.

1.4 Thesis Outline

The thesis is structured as follows.

- Chapter 2 reviews the state of the art, including some background for the main concepts used in the thesis. We introduce the general problem of software evolution. We provide a background on MASs and NMASs. We review the models and languages used in the areas of Requirements Engineering (RE) and NMAS to represent requirements and norms and to support their run-time monitoring. We outline the most common approaches and the core architectural choices for self-adaptation. Finally, we discuss the problem of norm and requirements revision and differentiate traditional model-driven approaches from data-driven approaches.

- Chapter 3 presents our supervision framework. We apply the framework to support the evolution (manual or automated) of requirements of socio-technical

systems. We present our mechanism to validate with run-time data a variety of assumptions that are made during the design of requirements models. We show how the validity of the assumptions can be used to guide the revision of requirements. In order to do so, we introduce the main component of our framework: a feedback loop that continuously monitors, evaluates and revises the requirements. We present some of the strategies that we propose for deciding how to revise requirements and norms based on data, and we use them to suggest and perform a revision of requirements models so to select appropriate alternative configurations of requirements.

- Chapter 4 focuses on the run-time data-driven revision of the sanctions used to enforce norms in a MAS. We leverage, in addition to run-time data about the behavior of agents, also some knowledge about their preferences. We focus on agents with rational preferences so to being able to effectively influence their decisions by means of sanctioning [202]. We use the information about their behavior and their preferences to determine, by means of a number of different revision strategies, new sanctions that are expected to improve the achievement of the system's objectives, if used to enforce the norms.

- Chapter 5 discusses how to concretely revise the components of conditional norms with deadlines. The revision that we study is based on a dataset of execution traces describing the behavior of the agents in the MAS, and it aims at improving the *alignment* between the norms and the system's objectives with respect to the dataset of traces.

- Chapter 6 draws the conclusions of the thesis and presents future directions.

Chapters 3-5 represent the technical contributions of the thesis. Each of them corresponds to a work published, or currently under review for publication, in an international journal in relation to this thesis. Since each of the publications is self-standing and includes an experimental evaluation, they are reported only with minor changes. For this reason, the reader may find some repetitions in the different chapters.

1.5 Publications

We list here published work related to this thesis.

International Journals

1. Davide Dell'Anna, Fabiano Dalpiaz, and Mehdi Dastani. "Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing". *Automated Software Engineering*, 26.3 (2019): 513-557.

2. Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. "Runtime revision of sanctions in normative multi-agent systems". *Autonomous Agents and Multi-Agent Systems*, 34.2 (2020): 1-54.

3. Davide Dell'Anna, Natasha Alechina, Brian Logan, Maarten Löffler, Fabiano Dalpiaz, and Mehdi Dastani. "Data-Driven Revision of Conditional Norms in Multi-Agent Systems". [*under review*]

International Conferences and Workshops

4. Davide Dell'Anna, Fabiano Dalpiaz, and Mehdi Dastani. "Reasoning about norm revision". In *Preproceedings of the 29th Benelux Conference on Artificial Intelligence*, BNAIC 2017, pp. 281 to 290

5. Davide Dell'Anna. "Requirements-driven supervision of socio-technical systems". In *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 23rd International Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ 2018.

6. Davide Dell'Anna, Fabiano Dalpiaz, and Mehdi Dastani. "Validating goal models via Bayesian networks". In *Proceedings of the 5th International Workshop on Artificial Intelligence for Requirements Engineering*, AIRE@RE 2018.

7. Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. "Runtime norm revision using Bayesian networks". In *Proceedings of the 21st International Conference on Principles and Practice of Multi-Agent Systems*, PRIMA 2018.

8. Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. "Runtime Revision of Norms and Sanctions based on Agent Preferences". In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS 2019

2 State of the Art

2.1 Introduction

Software systems are expected to operate in a constantly changing environment, and in domains that are only partially understood [205]. Change in the applications and in the domains in which they are applied is inevitable and constant. This follows from the fact that real-world software, and the environment in which it operates, have a potentially unbounded number of properties, forcing the system designer to explicitly or implicitly make *assumptions* that are then reflected in the requirements specification [191].

In his seminal work, Lehman [189] distinguishes three types of software, based on the extent to which the software is intertwined with the environment in which it operates, and therefore on the degree to which it is subject to change.

The type of software that is least subject to change is called *S-type*. The specification of S-type software is a formal definition of a problem for which there is a desired and correct solution. For example, a program that determines the lowest common multiple of two integers is an S-type software. S-type software is not affected by change in the environment, as its correctness and value only relate to its specification, which does not change unless the problem statement itself changes.

P-type software is one whose problem statement (or its solution) cannot be precise, but approximates the real-world situation. An example of P-type software is a chess game: due to the complexity of the problem, the software is forced to introduce approximations in order to be practically used. The type of approximation is determined by the designer, and reflects to some extent the designer's personal point of view and available knowledge. P-type software is more subject to change than S-type software. Even though the problem to be solved can be precisely defined, the value of a solution depends on the real world context in which it is embedded. When the context changes, the current solution to the problem may need to be changed as well.

Finally, *E-type* software is fully embedded in an environment, where it is expected to perform human or societal activities. An E-type software, however, is not only part of the environment it operates in, but it also actively affects it and influences it. A traffic control system is an example of E-type software: the behavior of the system is determined both by the software itself and by the way it is used by the user. Lehman

writes,

> "As they become familiar with a system whose design and attributes depend at least in part on user attitudes and practice before system installation, users will modify their behavior to minimize effort or maximize effectiveness. Inevitably this leads to pressure for system change."

E-type is clearly the most complex type of software to design. In this category fall a number of highly complex systems that we encounter in our daily lives. The *coalition of systems*, as defined by Sommerville *et al.* [258], that led to the infamous Flash Crash of 2010, when in about 10 minutes a number of stock indeces unexpectedly collapsed, causing the disappearance of trillions of dollars of market value, is an example of such type of system. The already mentioned *Socio-Technical Systems* (STSs) are another example of complex system falling in the E-type category. In a STS, continuous interactions take place between humans, machines and environmental factors [36]. Examples of STSs are a smart home that helps patients carrying out their daily activities; or a smart city [73]. These systems involve both technical and social entities, continuously interacting according to complex and ever-changing dynamics in a shared environment. Dynamism, uncertainty and change are main features of such systems. Because of these features, the assumptions underlying the design of an E-type software, are gradually invalidated in time, and this determines the need for its continual change and evolution [190].

In the context of E-type software, the environment in which software operates and the objectives and expectations of its users and stakeholders continuously co-evolve. In response to change, not only the current software implementation shall be checked against its requirements, but also the requirements themselves shall be evaluated, and revised if necessary, by considering the global behavior of the system. Belady and Lehman [37], as well as Harker [159], illustrate that changing requirements are more common than stable ones in software development and that requirements typically need to be redefined to new uses.

To design, maintain, and evolve the requirements of complex modern systems that are embedded in our society, requires to consider not only the individual software components, but also the interactions between its technical and social factors [36]. Traditional software and requirements engineering methods mainly focused on the technical aspects of systems [81], which simply ruled out by the system's specification the possible illegal behaviors [206]. These approaches are not easily suitable to describe and regulate the behavior of the social and weakly-controllable actors that are at the core of complex modern systems [258].

Recent software system's engineering approaches, however, have begun to move towards so-called *normative systems* [151, 264]. One of the most important aspects of normative systems, is that they allow to *explicitly* distinguish between the expected, or ideal, behavior of the system and the actual exhibited behavior. Jones and Sergot [169] have illustrated that computer systems can be seen as instances of normative systems, that is sets of interacting agents (human individuals, or computer systems, or collections of either of them) whose behavior can be governed by norms, which prescribe what agents should or can do. Chopra *et al.* employ norms (commitments) to

specify the interactions of a Socio-Technical System [79]. They introduce the so-called Interaction-Oriented Software Engineering (IOSE), a software engineering paradigm which expressly aims at dealing the social aspects of STSs. In particular, IOSE, instead of focusing on technical implementation, focuses on the engineering of social protocols. Social protocols specify how (social) relationships among the participants of the system by means of normative expectations such as commitments, authorizations, or prohibitions. In doing so, the IOSE paradigm promotes the openness of software systems and the autonomy of its principals.

Multi-Agent Systems (MASs) are a choice for the realization of modern complex systems that is aligned with the normative paradigm and has been proposed by several authors in the literature [30, 231, 256]. MASs are a type of systems where agents represent their stakeholders and autonomously act and interact on their behalf. This is also our point of view. In this thesis, we employ multi-agent systems as a technology for modeling, studying and supervising modern complex systems. In particular, in line with the IOSE paradigm, we focus on *normative multi-agent systems*, where *norms* are used as a means to coordinate the behavior of the agents in order to guarantee desired properties of the whole system. In the rest of the thesis we will refer to norms to talk about explicit requirements for the behavior of interacting agents in a computer system, and we will often use interchangeably the two terms.

In Sec. 2.2, we provide some background on MASs and describe the principal features of normative MASs. In Sec. 2.3 and 2.4, we overview the literature concerning frameworks for the run-time adaptation of software systems. We examine the best practices in the literature of self-adaptation at the requirements level, which identify the need for requirements to be kept alive at run-time in order to successfully monitor and analyze them, and employ feedback loop architectures to endow systems with continuous adaptation capabilities. In Sec. 2.5, we outline the work related to the revision of requirements or norms.

2.2 Multi-Agent Systems

In the field of Artificial Intelligence (AI), a Multi-Agent System (MAS) is a type of system that consists of a number of agents interacting with each other in a shared environment. *Agents* in MASs are computer systems characterized by their autonomy, that is, their ability to independently reason and take decisions about what to do in order to satisfy their design objectives [297]. Generally speaking, agents are self-interested components and base their decisions on their own *preferences* or desires, which typically reflect the preferences and objectives of the user they operate for [278]. Agent preferences are often associated to utility functions, so that they can be used as a rational criterion to choose between different alternatives, based on their expected outcomes [202, 286].

Different characterizations of agents have been proposed. Agents are typically characterized by the following properties [298], which aim at distinguishing them from classical programs:

- *autonomy*: their ability to operate without direct intervention of humans, having some control over their actions and internal state;

- *social ability*: their ability to interact with other agents or humans;
- *reactivity*: their ability to perceive the environment in which they operate so to timely react to the changes that occur in it;
- *pro-activeness*: their ability to take the initiative and exhibit behavior that is driven by their own goals and not only to respond to the environment.

Agents are often modeled and implemented by making use of intentional notions [108, 297]. The *belief-desire-intention* (BDI) model [62], for example, attributes to agents mental states such as beliefs, desires, intentions, and characterizes the agents deliberation and reasoning in terms of these mentalistic notions [253]. This model, whose underlying ideas have roots in philosophy [122], has been widely accepted and employed in AI and MASs to describe and guide the implementation of intelligent agents in a human-like fashion. IRMA (Intelligent Resource-Bounded Machine Architecture) [63] and PRS (Procedural Reasoning System) [150] are two notorious examples of BDI-based agent architectures. A variety of languages for programming BDI agents has also been proposed over the years, following the Agent Oriented Programming (AOP) paradigm introduced by Shoham [253]. Examples of such languages include AGENT-0, presented by Shoham himself, AgentSpeak(L) [233], 3APL [162], Jason [57], Jadex [230], GOAL [112], 2APL [104] and the 2APL Java-based library [105].

Thanks to their properties of concurrency and independence of their components, as well as their affinity with real-world and human aspects, MAS have been used both in academic research and in industry in various domain, including distributed systems, transportation, logistics, smart grid, as well as in the context of smart cities [70, 174, 219, 237].

In a MAS, successful interaction between agents depends on their ability to interact with each other and with the environment where they operate. The internals of an agent (e.g., their preferences, beliefs, or goals), however, are typically unknown to, and often not aligned with the ones of, other agents. In fact, agents in a MAS can be engineered by different organizations or parties with different design objectives so that, generally speaking, no common goal between different agents can be assumed [297]. Agents developed by different parties may be written in different programming languages and based on different architectures. This is often the case in the so called *open* MASs. As described by Davidson [111], in contrast to *closed* MASs, where the participating agents are known beforehand (i.e., when the system is being designed), in an open MAS different agents can enter or leave the system as they please, and therefore the agents participating into the system cannot be known beforehand, nor their interactions can be always fully predicted. Artikis and Pitt [22] identify as one of the fundamental properties of an open MAS its *neutrality* w.r.t. the internal architecture of their members, which need therefore to be treated as black boxes. Open MASs are particularly suitable to characterize and deal with the complexity of modern socio-technical systems, similarly involving a multitude of entities, including humans, whose internals are not fully known nor controllable.

Since the agents in an open MAS need to be treated as autonomous black boxes, leaving the agents interacting freely without any form of external control makes it hard to predict and steer the emergent behavior of the system, and may lead to undesired behaviors [139, 166].

2.2.1 Normative Multi-Agent Systems

Inspired by society, where interactions are affected and regulated by our social abilities, researchers have adopted normative and organizational approaches to MASs. This led to the *Normative Multi-Agent Systems* (NMASs), where agents must take into account social aspects, such as roles or norms, when interacting with each other [75, 283]. These approaches are suitable for systems that need to support real-world organizations (e.g., an online auction system, or a conference management system), since the organizational structure can be reflected in the system itself thereby reducing the conceptual distance between the software and the real-world [305]. Moreover, NMASs are considered promising models also for dealing with the complexity and dynamics of complex systems [50]. An extensive and detailed overview of the variegate research that is being conducted in NMASs can be found in the Handbook of Normative Multi-Agent Systems [78].

Balke *et al.* [31] define a normative multi-agent system as follows.

> "A normative multi-agent system is a multi-agent system organized by means of mechanisms to represent, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment."

Norms and their classifications

Norms are of central importance in normative multi-agent systems. Norms have been studied in a variety of different fields besides computer science, including sociology [130, 153], philosophy [173, 287], economics [131, 218], as they are a fundamental technique in our society for coordinating the activities of individuals [194]. Due to their multi-disciplinary nature, there is no standard definition of norms and different classifications have been proposed [20]. Gibbs [153] identifies in the sociology literature nineteen types of norms and provides the following general description of norm.

> "A norm in the generic sense involves: (1) a collective evaluation of behavior in terms of what it *ought* to be; (2) a collective expectation as to what behavior *will be*; and/or (3) particular *reactions* to behavior, including attempts to apply sanctions or otherwise induce a particular kind of conduct."

We consider here three particular classifications that will help us characterizing the types of norms that we will consider in the following chapters.

Starting from Searle's theory of the construction of social reality [247], one classification of norms is provided by Boella and van der Torre [49], and distinguishes *regulative* and *constitutive* norms.

- Regulative norms describe the behavior that is expected from agents. Typically they are described via deontic statements of obligation, prohibition and permission that regulate the possible actions and behaviors of the agents in certain operating conditions. An example of regulative norm is a norm *cars are obliged to have a speed below 80 km/h.*

- Constitutive norms are the rules that determine what *count as* what, in a given society. They are meant to create *institutional facts*, i.e., facts that give a meaning to *brute facts* in the context of a normative system [247]. An example of constitutive norm is *raising a hand during an auction counts as making a bid*, which states that *raising a hand*, a brute fact, has the meaning of *making a bid*, an institutional fact, if done during an auction.

In this work, we focus on *regulative norms* and we assume norms to be formal specifications of deontic statements that impose prohibitions, obligations, or permissions on the behavior of agents.

A different classification of norms is provided by Coleman [91] with respect to the way the norms emerge and are enforced. Coleman distinguishes *conventions* from *essential norms*.

- Conventions are norms that emerge naturally within a population of individuals without an explicit enforcement system. Conventions are associated to the so-called *social norms*, or *s-norms* [270], and are seen as regularities of behavior that spontaneously emerge from the mutual beliefs and expectations of agents [194]. A classical example of convention is the agreement of driving on the right-side of the road. Social norms relate to an interactionist view on normative multi-agent systems, i.e., a bottom-up view where norms are not enforced by an authority (or *institution*), but agents are discouraged from violating them by the possibility of social blame or exclusion from the group of agents they are part of, or with whom they share similar values [45].

- Essential norms, conversely, are associated to *rules*, or regulative norms (*r-norms*). They are created and enforced by an authority with the purpose of controlling and coordinating the behavior of the agents when there is a conflict between the individual goals of self-interested agents and the global goals of the system. Regulative norms relate to a legalistic (sometimes essential norms are called *legal norms*) view on normative multi-agent systems, i.e., a top-down view where the normative system is meant to regulate the emerging behavior of (possibly open) systems without over-constraining the autonomy of the participating agents. In such a view, norms are sometimes associated also to explicit *sanctions*, which are measures (e.g., a fine) that discourage agents from violating them. A speed limit (regulative) norm, for example, may be enforced by an institution in order to improve the safety of the road. Sanctions make sure that selfish agents obey the norm.

In this dissertation, we adopt an exogenous approach for the coordination of agents in a MAS [68]. As opposed to an endogenous approach that assumes that norms are internalized by the agents, an exogenous approach is agnostic about norm internalization. The focus of this dissertation is on mechanisms aimed at regulating the behavior of heterogeneous agents that do not necessary share the same objectives with other agents and with the system designers and stakeholders, and whose internals may be unknown. With respect to Coleman's classification, we focus on *essential norms* that are enforced by an institution.

The main focus of this work, therefore, is on an external authority (institution) in a MAS that is aimed at enforcing norms in order to coordinate the behavior of the agents. Engineering such coordination mechanisms by means of an external authority also supports the general principle of 'separation of concerns', and supports open MAS as it does not over-constraints the implementation of the participating agents [304].

Observe that an exogenous approach does not imply that agents do not internalize norms, nor that new norms cannot emerge within a society of agents. Similarly to human society, norms that are enforced by an exogenous authority may be internalized by agents and become part of their beliefs and internals (e.g., stopping at a red light). Vice-versa, norms that emerge from the behavior of the agents may become laws and be enforced by an institution if detected, and their explicit regulation may improve the compliance and their effectiveness (e.g., driving on the right) [221]. In this dissertation, while we adopt a top-down approach, focusing on an institution that aims at enforcing norms, we support, but we do not explicitly focus on, the internalization of norms and their emergence in the population [5, 95, 242]. Moreover, in our framework, the institution makes extensive use of execution data, obtained from the exhibited behavior of the agents, in order to critically evaluate and decide how to revise the enforced norms. Analyzing behavioral data allows the institution to take into account the agents' preferences and choices in its decision mechanisms. To give an example, in our framework, if some agents decide to violate a norm, and the violations appear not to be harmful for the system-level objectives, our institution will tend to accommodate such violations by switching to less restrictive norms.

Finally, a third classification of norms is that recently provided by Alechina *et al.* [12]. This classification is based on whether norms and their violations are specified in terms of states, actions, or behaviors.

- State-based norms specify (properties of) states of the system that are prohibited or required. A state typically characterizes a state of affair of the system. For example a state-based norm is *a car is obliged to have an insurance.*
- Action-based norms (also called transition-based, or event-based, norms) specify prohibited or required transitions between states of the system, that can be due to actions performed by individual agents, but also actions performed by groups of agents. An example of action-based norm is *it is prohibited to enter the road.*
- Behavior-based norms (also called path-based norms) specify prohibited or required behaviors (i.e., temporal patterns of states) of the agents or of the system as a whole. Among this class of norms, we find conditional norms with deadlines, which express states that are prohibited, or required, between states where a condition holds and states where a deadline holds. An example of this type of norm is *after entering the city center the car is prohibited to speed over 50 km/h until it exits the city center.*

In this dissertation, we will first devise general strategies that suggest a type of revision required of a norm based on data acquired during the execution of the system (for example we will devise strategies that suggest to relax, or weaken, a norm, when it appears from data that the norm is too restrictive). In discussing those strategies, for example in Chapter 3, we will abstract from the specific type of norm, as we

consider the suggestion independent from the type of norm. In Chapter 4, we will focus on *sanctions*, which can be used as an enforcing mechanism for different types of norms. In Chapter 5, instead, we will focus on how to specifically revise a norm, given a type of suggestion. In that case we will consider behavior-based norms, and in particular conditional norms with deadlines.

Norm-aware agents

In order for agents in a MAS to be able to choose whether to obey to norms or not, agents should have an explicit knowledge of the norms and should be able to reason about them. Conte *et al.* [75] describe requirements for an agent to be considered *norm-autonomous* (sometimes referred to as *norm-aware* or *normative*). A norm-aware agent, among other things, is able to recognise that a norm exists; it is able to accept it, i.e., to determine if a norm concerns its case or not; and it is able to deliberately follow it or violate it.

Many architectures and solutions for norm-aware agents have been proposed in the literature to represent, reason and make decisions about norms. Deontic logic [287] has been commonly used to represent and reason about obligations, prohibitions and permissions [93], and it has been extended in several ways to support for example the specification of deadlines [124] or to express the actors who initiate actions or choices [296].

Following the success of the BDI model, BDI-based architectures have been proposed to embed concepts such as norms and regulations in the deliberation process of agents [67]. For example, Castelfranchi [74] represents norms as mental objects entering the mental processing, that interact with beliefs, goals and plans. Norms have impact on goals generation and goal selection, but also on plan generation and plan selection. Dignum *et al.* [125] discuss how to integrate deontic events as normative beliefs in the original BDI framework in the context of social agents.

The BOID (Beliefs-Obligations-Intentions-Desires) architecture [65, 106] introduces a normative aspect to the BDI architecture, allowing agents to choose to comply with obligations. Different types of agents are defined, based on their preference relation over their components. For instance, a *selfish* agent gives priority to its desires over obligations; a *simple-minded* agent priorities intentions over desires and obligations; and a *social* agent gives more priority to obligations than desires. Preferences allow agents to solve the conflicts that are intrinsically carried by the introduction of norms in agents' reasoning [66, 183]. Kollingbaum *et al.* [183] describe different levels of inconsistency that can occur during the adoption of a norm from an agent. They discuss conflicts between norms in the context of the Normative Agent Architecture (NoA) [182], where norms are used as a filter to remove plans that would result in violation.

Preference ordering over goals and norms are common to different architecture as a means to solve conflicts between them. In N-2APL [8], agents associate an ordinal number to goals and sanctions (that would result from the violations of norms) that represents the importance of achieving a goal and avoiding a sanction respectively. The importance is then used by the agent to decide whether or not to drop a goal or incur in a sanction.

A number of programming languages for normative agents have been proposed that extend the languages for non-normative agents. N-2APL [8] and N-Jason [188] are two of them that follow the AOP approach.

In this dissertation, we focus on an exogenous institution which has the task of enforcing norms and monitoring and handling their violations. We will abstract from the specific underlying reasoning schemes of individual agents, which, in line with the fundamental property of neutrality of an open MAS, we will assume not to be known to the institution. We will assume, however, that agents are norm-aware, in the sense that they respond to the enforcement of a norm. In Chapter 4, we will consider a case where agents act rationally according to their preferences and the institution has some knowledge about such preferences. As the focus of this thesis is mainly on mechanisms of revision of the norms, we also do not explicitly consider the *roles* that agents can have within an organization, and the aspects that organization-aware agents shall take into account when making decisions [279]. Limitations concerning these assumptions will be discussed in the corresponding chapters and in the conclusions of this thesis.

2.2.2 Norm Enforcement and Monitoring

Being able to adopt norms in a MAS requires an implementation of a mechanism of norm enforcement and monitoring that is responsible for detecting violations of the norms and handling these violations [301].

The enforcement of a norm depends on the level of control that the institution has over the agents and the verifiability of the content of the norms [282].

Vasquez *et al.* [282] provide an orthogonal classification of norm to the ones illustrated in Sec. 2.2.1, and identify three types of norms with respect to their verifiability.

- Computationally verifiable norms: they can be verified at any moment by the institution enforcing it. An example of this type of norm is *no car shall enter the road*.
- Non-computationally verifiable norms: they can be verified by the institution but it is computationally hard to do so, or additional human intervention is required. For example a norm that prohibits to publish fake news in a social network may be hard to verify from an institution.
- Non-verifiable norms: they cannot be verified by the institution because they are not observable. For example an institution cannot verify if an agent has certain beliefs.

In this thesis, we will mostly focus on computationally verifiable norms. We will also consider, however, system's objectives which concern what the stakeholders of the system (represented by the institution) desires to achieve by means of enforcing (computationally verifiable) norms on the agents. In this sense, some of such system-level objectives could be associated to non-computationally verifiable (higher level) norms. For example if one of the system's objectives is to guarantee the satisfaction of the (human) users of the system, such objective is non-computationally verifiable, as it needs a human feedback concerning their satisfaction. We distinguish therefore (computational) verifiability, which we assume for the norms, from the notion of

measurability (i.e., the capability of obtaining a quantitative measure of a variable, either computationally or non-computationally), which instead we assume for system's objectives.

Besides their verifiability, an institution also needs to enforce in some way the norms on the agents. Two ways of enforcing a norm are mainly distinguished in the literature: *regimentation* and *sanctioning* [109].

- Regimentation is based on the idea of preventing violations to occur. For example, in a smart traffic system, a regimentation strategy could be to close a road to prevent cars from entering the road. Regimentation assumes that the system can somehow annul a violating action before it actually takes place. This does not assume control over the internals of the agents: the agent can still decide and try to perform a certain action, but the result will simply not be executed by the environment.

- Sanctioning, instead, is based on the idea of reacting to the violation of the norms. An agent's decision to obey or violate a norm, is motivated, or discouraged, by certain rewards or sanctions. Even though also regimentation could be considered as a sort of punishment (i.e., sanction), sanctioning is considered granting more autonomy to the agents, as agents technically can still perform the actions [7]. In the smart traffic system example, a sanctioning strategy could be to impose fines on cars that do enter the road.

In this thesis, we investigate both regimentation and sanctioning, and we will propose techniques to revise both the norms and their sanctions. We do not explicitly focus on rewards, or incentives, for the agents to perform a certain behavior [123]. We note, however, that sanctions can be also seen as a positive incentive to motivate the agents to comply with the norms, when compared to the negative reward the agent would receive if violating the norm. This is also in line with the idea of sanctions as a deterrence to discourage agents to perform illegal behaviors [7].

Practical implementations of norms and norm enforcement

Several practical implementation of norms and their enforcement have appeared in the literature in recent years, often adopting the so-called Organisation Oriented Programming (OOP) approach that aims at implementing an exogenous organization aimed at constraining, by means of norms, the behavior of the agents participating in the system. One of such approaches is ISLANDER/AMELI [135, 136], which combines ISLANDER, a modelling language for specifying institutions in terms of rules and norms, with AMELI, a platform that implements the infrastructure that allows agents to interact and communicate, and the institution to enforce the norms. In ISLANDER/AMELI, norms are regimented and action-based. Another related framework is $\mathcal{M}oise^+$ [163], which allows to specify multi-agent systems distinguishing the structural, functional, and deontic organisational dimensions of the system. The deontic dimension, in particular, concerns concepts such as obligations and prohibitions and it is the dimension that we mainly consider in this thesis. In the extension $\mathcal{S}-\mathcal{M}oise^+$ [164], agents are given access to the current state of the organization

and they are allowed to change the organization and its structure, as long as (regimented) norms prescribing prohibited or required states are not violated. Tinnemeier *et al.* [267] present a norm-based programming language aimed at implementing exogenous institutions that control the behavior of agents by means of (state-based) norms. An associated interpreter, called 2OPL [109], has the task of monitoring the behavior of the agents and enforcing the norms when necessary. Norms, in such framework, can be either regimented or enforced by means of sanctioning and they come with an operational semantics, so that verification techniques can be used to verify executions against these norms. Finally, some frameworks combine different approaches. For example, in JaCaMo [54] an organisational artifact is implemented with the $\mathcal{M}oise^{+}$ specification, and norms are enforced by means of sanctioning, which is delegated to organizational agents.

2.3 Designing Software Systems

In designing a software system, the process consisting in the elicitation, specification, verification and validation, and management of the requirements of the system is called requirements engineering. The purpose of requirements engineering is to acquire sufficient knowledge about the problem domain, and to elicit requirements from which to derive a specification that will meet the needs of the stakeholders [275]. Requirements are of foremost importance in software engineering, for they are the design artifact that will drive the software implementation by carrying information about what are the objectives of the software [134]. Zave and Jackson [306] first formulated the requirements problem as follows.

> *Requirements Problem.* Given the requirements R (optative statements of desire) and the domain knowledge W (properties and assumptions about the world the software must operate in), find a specification S such that $W, S \vdash R$.

Given the domain knowledge W, the specification S of the system is expected to completely cover the system's requirements R and to be correct and consistent in its technical definition. The specification is then used to guide the implementation of the software.

Such formulation of the requirements problem was later revised and extended by Jureta *et al.* [170] to overcome some of its limitations when applied in practice. Jureta and colleagues propose a new core ontology, whose concepts are mapped to the DOLCE foundational ontology [203]. They associate the Zave and Jackson's domain assumption, requirement, and specification concepts to propositional attitudes, i.e., belief, desires, and intentions, respectively. They characterize Functional Requirements (FR), which describe what the system must do, and Non Functional Requirements (NFR) [216], which describe desired qualities of the system and their desired values[1], with the notions of *goals* and *quality constraints*, respectively. They also characterize more abstract NFR, which refer to desired non-verifiable qualities,

[1]This is in line with the recent Requirements Engineering research literature, where NFRs are seen as requirements over qualities [155, 195].

such as "high convenience", with the notion of *soft-goals*. In order to obtain an evaluation of soft-goals in practice, the referred quality can be *approximated* with a quality of a quality constraint. Because of this approximation, the authors use of a non-monotonic operator $\mathrel{|\!\sim}$, instead of $\vdash$, to describe the satisfaction relation in the requirements problem. The specification, finally, is characterized with the notion of *plans*, whose execution is supposed to bring about states of the world in which the goals and all quality constraints (including those that approximate the soft-goals) are satisfied.

In the context of enterprise architectures, Greefhorst and Proper [157] introduce the concept of architecture principles as means for the system's stakeholders to express a vision of the system's objectives. They notice how architecture principles can be seen as *credos*, normative principles (declarative statements that normatively prescribe properties of the system) expressing a fundamental desired belief. Similarly to Jureta's concept of soft-goals, credos are not yet specific enough to actually be concretely enforced. In order to use these principle to guide the design of the system, they need to be made more specific, so that it is possible to assess their compliance. Greefhorst and Proper claim that once credos have been reformulated specifically enough to be measurable, they can be referred as *norms*.

In this thesis, we make use of the concepts introduced by Jureta *et al.*, and take a similar perspective to Greefhorst and Proper. In particular, we focus on the revision of computationally verifiable (as per Sec. 2.2.2) requirements, which we associate with goals and quality constraints. We associate system's objectives, i.e., non-computationally verifiable higher level requirements, with soft-goals. Since we consider requirements as norms aimed at regulating the behavior of autonomous weakly controllable components of the system (agents) whose internals are unknown, we will not assume that the intentions of such agents, i.e., their plans, will be in line with the system's requirements, nor that they will aim at satisfying the requirements.

2.3.1 Design-Time Engineering of Requirements and Norms

In the framework proposed in this dissertation, we consider the output of a *design-time* phase as (part of) the input for the revision process, which we see as a *run-time* phase. To make it possible, or to facilitate, the run-time revision of the requirements of a system, however, the prospect that requirements could be revised at run-time must be at least partly taken into account already at design-time, for example by choosing appropriate languages or models to represent requirements or norms.

Ernst and colleagues [134], for example, illustrate the importance of *well-structured tangible requirement artifacts* that represent all the aspects of the requirements problem and capture stakeholders' objectives, domain assumptions and implementation options. This is particularly true for systems that are characterized by their continuous evolution, such as E-type software. The reason is that, in the context of *requirements evolution*, the availability of such artifacts (or models) allows for their monitoring and for reasoning about them at run-time.

We discuss here design-time approaches for the engineering of norms and requirements. Then, we point out some limitations and extensions of these approaches for supporting the run-time evolution of software systems.

Requirements engineering methodologies and requirements modeling languages

From a software engineering perspective, in the last years, requirements engineering approaches, and especially those aimed at supporting the run-time analysis and evolution of requirements, started representing requirements via requirements models [40]. Requirements models [271] can be created with different purposes, including specifying a system, supporting testing, or simply for increasing clarity about the system. Depending on the purpose, the analysts may decide to represent different aspects of the system under development and its environment, such as the information structure, scenarios, goals, objectives, or requirements. Requirements models, and in particular hierarchical requirements models, organize the requirements for a system as refinement trees, where high-level objectives are refined in terms of more specific requirements and system functions.

Within the landscape of requirements modeling languages, goal models have been widely used in the literature [101, 275, 303]. The term Goal-Oriented Requirements Engineering (GORE) describes an approach to requirements engineering where the needs of the stakeholders of a system are represented as goals, in line with the ontology proposed by Jureta *et al.*. Goal models are graphs where each node can represent a goal, an action, an agent, an entity, or an event, and edges represent semantic links between such elements. In goal models, goals are typically organized in hierarchies via AND- and OR-*decomposition* links. Sub-goals of a goal are assumed to be necessary in order to satisfy the main goal. Goals can have positive or negative impact on soft-goals, non-functional requirements whose satisfaction does not depend on a clear cut criteria, e.g., *minimize* CO_2 *emissions*.

The concept of goal model is considered central in the KAOS (Knowledge Acquisition in autOmated Specification) methodology [102], which is meant to support the requirements acquisition phase, where designers try to understand which are the requirements of the system to-be.

Goals are used also in the agent-oriented modelling framework iStar (or i*), first introduced by Yu [302], and later consolidated by Dalpiaz *et al.* [101], which is used in requirements engineering and organizational modelling to model the stakeholders of the system and their objectives and relationships, and to evaluate alternative ways to construct the new system. A crucial concept in iStar, is the one of *actors*, which can represent concrete agents with specific capabilities, but also more abstract agents representing *roles* and responsibilities, or even positions in the organization. Actors in iStar have goals that aim to achieve, and depend on each other for the fulfilment of such goals. Along the same lines, agent-oriented software engineering (AOSE) methodologies such as Tropos [64] have been proposed in the literature. Tropos, for instance, imposes some additional constraint to iStar so to employ mentalistic notions such as goals, plans and tasks through five phases of software engineering, i.e., from the early requirements analysis to the implementation.

Formal approaches

A number of complementary approaches, based on formal methods, have been proposed alongside the requirements (and software) engineering methodologies outlined above. Formal approaches are used to formally model the systems under development,

so to provide formal guarantees about their behavior and their properties when satisfying the given requirements. The requirements of the system in these approaches are typically represented in some form of logic, such as propositional logic [277], propositional dynamic logic [109], temporal logic [35, 252, 278], or simply by means of a set of states or actions that are considered undesired and should not occur. A formal model that describes all possible aspects of the system is then checked to prove that the requirements are satisfied.

Classical formal methods employ transition systems (directed graphs, whose nodes represent states of the system and edges represent state transitions) or automata to model the possible behaviors of a system. Model checking techniques [84] are employed to verify that the model satisfies certain properties that can be defined for example in Computational Tree Logic (CTL) [83]. If that is not the case (for example if the transition system contains some paths that lead to undesired system states) the model needs to be changed (for example by removing or preventing certain transitions between states) until a satisfactory solution is found.

Concurrent Game Structures (CGS) have been considered to deal with open systems, whose behavior depends both on the system and on the environment in which it operates. In a CGS, the state transitions result from a profile of choices of actions, one for each agent in the system, including also the environment. In these cases properties to be checked are typically expressed in Alternating-time Temporal Logic (ATL) [19].

Formal approaches are particularly commonly used in the (normative) MAS research community, often taking into account game theoretical concepts in order to study the properties of a MAS and to design opportune norms. Agotnes *et al.* [4], for example, represent multi-agent systems as Kripke structures, a variation of (labelled) transition systems where labels are given to states instead of transitions, and describe a normative system as a subset of such structure. Introducing a norm in a multi-agent system, corresponds therefore to removing some (bad) transitions from the Kripke structure. They represent agents' goals as ordered lists of CTL formulae and show that the utility of agents is higher when they decide to comply with the norms. Bulling *et al.* [68] apply methods from mechanism design to determine and verify if in a MAS, described as a CGS, a set of norms and sanctions is sufficient to motivate agents to act in the way desired by the system's designer, given the rational behavior of agents based on their preferences.

Onn and Tennenholtz [220] represent a MAS as a graph, and reduce the problem of designing social laws to a graph routing problem. They use graph theoretic methods to automatically compute a subgraph that will guarantee agents to achieve their goals regardless of the behavior of other agents. Along the same lines, Christelis *et al.* [82] propose a mechanism to synthesize norms that prohibit access to a set of predefined undesirable states. They employ local search algorithm to travers the state space and to identify so-called conflict free runs that are then used to produce action-based norms that prohibit to perform actions that lead to conflicting states.

Agotnes *et al.* [3] model the problem of designing a social law as a multi-objectives optimisation problem, and formulate the problem as an integer linear program. Similarly, Lopez-Sanchez *et al.* [197] show how to encode the problem of deciding which norms to enforce from a set of existing possible ones as a linear optimization con-

straint problem. They define a problem called Maximum Norm System Problem with Limited Budget which determines which subset of a predefined set of norms should be enforced. They encode in the problem definition also the cost of enforcing a norm and the relationships that may exist between them, such as exclusivity or substitutability.

2.3.2 Design-Time Assumptions

Traditionally, the development of software systems clearly distinguished the design phase and the run-time execution of the system. Design-time approaches like the ones described above, however, inevitably rely on a number of *assumptions* about the correctness of the requirements, about the system, and about the agents operating therein. For instance, formal approaches assume that all possible interactions that can occur between agents are known and can be taken into account in the model at design-time; or that once a norm is proven to guarantee the desired properties, that will be the case also once the system is put in place.

Boness *et al.* [55, 56] identify assumptions–in particular, assumptions regarding what is expected from the environment in which the system will operate–as one of four risk factors characterizing the potential for error and loss in software development that is acquired during the requirements engineering phase. They propose to explicitly represent assumptions when defining requirements within a goal oriented framework. They use the concept of assumption to help the system designers to assess, during requirement analysis, the confidence in (and the risk due to) the set of elicited requirements.

Ali *et al.* [15], identify a set of assumptions that are made by the requirement engineers based on their beliefs and knowledge about the system under design and its environment, and that are implicitly contained in the structure of requirements model. Some examples of such assumptions include:

- *Activation assumptions*, concerning the assumption that the description of the context in which a certain requirement should be activated (e.g., a requirement "the product shall be promoted to the customer" should be activated when "the customer is in the product area") are sufficient to cover all and only the cases when the requirement should be activated. For instance, if the customer has already entered the product area twice, wouldn't promoting the product every time risk to lead to a negative customer reaction?

- *Adoptability assumptions*, concerning the assumption that a certain software implementation guarantees the achievement of a requirement. For instance, is it always true that sending a message to the customer showing the features of a product guarantees the achievement of the requirement in the previous example?

- *Requirements achievement assumptions*, concerning the assumption that when given conditions are met, a certain requirement is satisfied. Is it sufficient for the customer to investigate the product to consider the requirement in the above example met?

Invalid design-time assumptions have been identified as one the main causes for software evolution [15, 191]. This is particularly true for complex modern systems, where the autonomy of the participating entities and the volatility of the environment and of the operating context lead to high run-time uncertainty [16, 294]. The notion of *context* becomes of foremost importance for this type of systems, as the context heavily influences the requirements, and changes in the context affect the validity of the assumptions.

Several works have discussed the importance of keeping track of the relationship between context and requirements at run-time [16, 144], as well as the need to explicitly consider assumptions made at design-time and continuously monitor them at run-time to being able to have requirements and software that reflect the reality.

In this dissertation, and in particular in Chapter 3, we propose strategies to use run-time data to statistically validate a variety of assumptions in a requirements model (including the ones presented by Ali *et al.* [15]) in different operating contexts.

2.3.3 Requirements and Assumptions at Run-time

In recent years, it has been shown that requirements and requirements models, traditionally used for design, implementation and verification purposes, can (and should) be used also at run-time to monitor and analyze the behavior of the system [48]. Doing so enables the collection of system execution data and its analysis, providing the essential inputs for detecting whether and when design-time assumptions become invalid [15] and to support system evolution or self-adaptation [243]. Baresi and Ghezzi [33] discuss the need of breaking the boundary between design-time and run-time. They illustrate that models used during software development should continue to live at run-time in order to cope, and evolve together, with the continuous evolution of the system operating environment.

Bencomo *et al.* [43] write,

> "Software models also have the potential to be used at runtime, to monitor and verify particular aspects of runtime behavior, and to implement self-* capabilities (e.g., adaptation technologies used in self-healing, self-managing, self-optimizing systems). A key benefit of using models at runtime is that they can provide a richer semantic base for runtime decision-making related to runtime system concerns associated with autonomic and adaptive systems."

The idea of endowing systems with self-* capabilities, comes from the fact that, especially in modern software systems such as the Internet of Things (IoT) or cloud computing, halting the system to maintain it in case of non-compliance with the requirements, or in case the assumptions made at design-time become invalid, is hardly an option, and often too costly [34, 196].

A core issue for the run-time maintenance of requirements, as we will see in Sec. 2.4, is that of *monitoring*. Requirements should be monitorable and verifiable during the system's execution. This is in line with the type of norms identified by Vasquez as computationally verifiable norms, as reported in Sec. 2.2.2. Doing so allows their run-time analysis and it allows to verify with execution data whether the assumptions

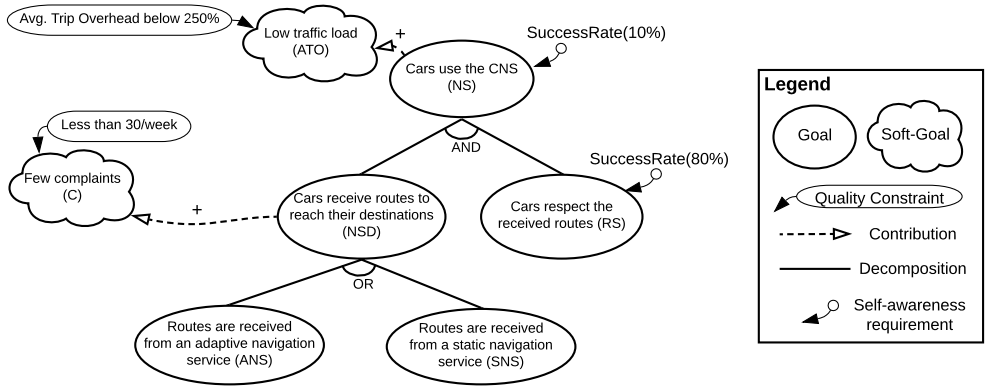


Figure 2.1: *Illustrative simplified goal model with awareness requirements.*

made at design-time are valid, and to possibly trigger a revision of both requirements and assumptions.

Goal and requirements models at run-time

Over the years, goal and requirements models have been extended in a number of ways to support their *runtime monitoring*. Dalpiaz *et al.* [99], for example, introduce Run-time Goal Models (RGM), a refinement of (Design-Time) Goal Models that introduces additional behavioral constraints about how goals are to be achieved (e.g., constraints on the ordering for pursuing sub-goals of a goal). They propose RGMs as an artifact to determine at run-time whether system operations are in accordance with its requirements.

Ali *et al.* [13, 14], describe Contextual Goal Models, which allow to specify a reference context to different elements of the a goal model. This allows to consider at run-time opportune parts of a goal model based on the current operating context. In a later work [15] they show how to use such contextual goal model at run-time to collect experience about the effectiveness of requirements and about the validity of the assumptions in different contexts.

Souza *et al.* [260] introduced the concept of *awareness requirements* (*AwReq*s) as meta-requirements that can be used to guide the run-time adaptation of systems. Awareness requirements are requirements about the run-time success or failure of other requirements, but also about the truth or falsity of (design-time) domain assumptions. *AwReq*s represent a type of annotation that helps qualifying the satisfaction of different criteria allowing for runtime monitoring of requirements (and assumptions).

To clarify these concepts, Fig. 2.1 shows an illustrative example of a simplified goal model, integrated with some awareness requirements. The goal model refers to a scenario where the city council of a smart city aims at improving the urban traffic by offering a Central Navigation Service (*CNS*). A major goal is identified: *at least 10% of the cars in the city shall always use the offered CNS* (*NS*). To satisfy this goal, two sub-goals are assumed to be necessary: *whenever a car starts a trip toward a*

destination, the car shall receive a route from the Central Navigation Service (*NSD*) and *at least 80% of all the route suggestions given by the CNS is respected by all the cars equipped with the CNS* (*RS*). The *NSD* goal can be met by either employing a dynamic navigation service (*ANS*) (for example a navigation service that adapts its suggestions to the traffic load and learns from its experience) or a static navigation service (*SNS*). In other words, *whenever a car equipped with the CNS starts a trip toward a destination, the car shall receive a route from an adaptive (static) navigation service.* These goals are *assumed* to help achieve two soft-goals concerning the global traffic load in the city (evaluated in terms of cars' average trip overhead) and the satisfaction of the users when using the navigation service (evaluated in terms of number of complaints).

Note that goals, as mentioned earlier, are organized in hierarchies via AND- and OR-*decomposition* links. For example, to achieve the goal *NS*, both goals *NSD* and *RS* shall be satisfied. OR-*decompositions* describe possible *exclusive* ways to achieve a goal. The expected positive or negative impact of goals on soft-goals is represented via *contribution* links, shown as dashed directed arrows. Goals that are not associated to an *AwReq* in Fig. 2.1 are required to be satisfied by every instance of the referred goal (e.g., an instance of goal *ANS* is created at run-time every time a car equipped with the CNS starts a trip, and such instance is satisfied if the car receives a route from an adaptive navigation service). These requirements are called *regular AwReq*s by Souza *et al.* [260]. Requirements like *SuccessRate*, instead, are called *aggregate AwReq*s: the satisfaction of the associated goal is determined in terms of groups of instances of the goal (e.g. an instance of the goal *NS* is created and evaluated for every car driving in the city, *NS* however is achieved if 10% of such instances is satisfied).

Goal models have been proposed as run-time artifacts also in the context of multi-agent systems. For example, Morandini *et al.* [214] describe an operational semantics of goals in goal models at run-time. This allows to use goal models also in more formal agent-oriented programming settings.

Requirements languages for run-time system adaptation

Besides requirements models, also languages for specific requirements have been proposed in light of a run-time system adaptation.

RELAX [294], for example, is a fuzzy logic-based requirements language for self-adaptive systems that allows to specify, with adequate operators, relaxed versions of a requirement during the requirement elicitation phase. The language is useful to specify requirements, and their possible variations, so to facilitate run-time approaches to their adaptation. For example, an *AS EARLY AS POSSIBLE* operator allows to specify a relaxed version of a requirement which can be enabled at run-time to provide the system with less strict (and fuzzy) constraints about when the requirement must be met.

In the normative MAS research community, *conditional norms* with *sanctions* and *deadlines* have been commonly used to express desired behavioral properties [9, 267]. A conditional norm is a tuple $(c, Z(\phi), d, s)$, where c, ϕ and d are boolean combina-

tions of propositional variables from a propositional language L, s is a propositional atom, and Z can be either O (indicating an *obligation*) or P (indicating a *prohibition*). c represents the condition that must be satisfied in a state in order to *detach* the norm. A detached norm persists as long as it is not obeyed or violated or the deadline d is not reached (a state where d holds is encountered). If the norm is violated, a sanction s is associated to the violation. Alechina *et al.* [10] show that one of the advantages of this type of norms is to provide a reasonable compromise between expressiveness and ease of reasoning. In particular they show that the semantics of the violation of conditional norms can be expressed in Propositional Linear-time Temporal Logic (PLTL), which is a known fragment of LTL, and they illustrate that the runtime monitoring for violations of conditional norms requires time linear in the size of the trace and constant space.

In this dissertation, we will make use of both run-time requirements models, and conditional norms with sanctions and deadlines. Both run-time requirements models and conditional norms are characterized by features that make them suitable and interesting for their run-time revision. In particular they both are *tractable*, in the sense that, as we have shown, they can be easily kept alive and monitored at run-time. They are *expressive*, as they allow a hierarchical composition and the characterization of relatively complex *behaviors*, and at the same time conceptually they are also relatively easy to understand. Conditional norms, for example, allow to express a variety of temporal patterns of behaviors, but they are also often encountered in our daily life. Think for example of a norm *after entering the city center, cars are obliged to have a speed below 50 km/h, until they exit the city center*. Finally, they are both *formal*, so that they can be interpreted and automatically revised mechanically, but they also allow to express properties with a certain degree of flexibility, typical of many real-world software engineering applications.

Concerning requirements models, we will not focus on a particular modeling language. We will consider, instead, a general notation of hierarchical requirements models extended with awareness requirements such as the one illustrated in Fig. 2.1, that will allow us to organize the requirements for a system as refinement trees, and to monitor their satisfaction at run-time. We will study then how to validate, at run-time and by means of a *statistical analysis of execution data*, the assumptions that underlay the structure of the model and its requirements, so to trigger a revision of the requirements, when needed. We will devise a number of strategies that suggest types of revisions of requirements based on the outcome of the assumption validation, and use these suggestions to perform a revision of the requirements. We will show how to apply the suggestions directly to revising the requirements model, in particular to selecting a new variant of the model that satisfies the given suggestions. Moreover, we will also consider specific conditional norms and we will show how to revise all their components (i.e., condition, regulated state ϕ, deadline, and sanction) so to synthesize new norms that are in line with the suggested revision.

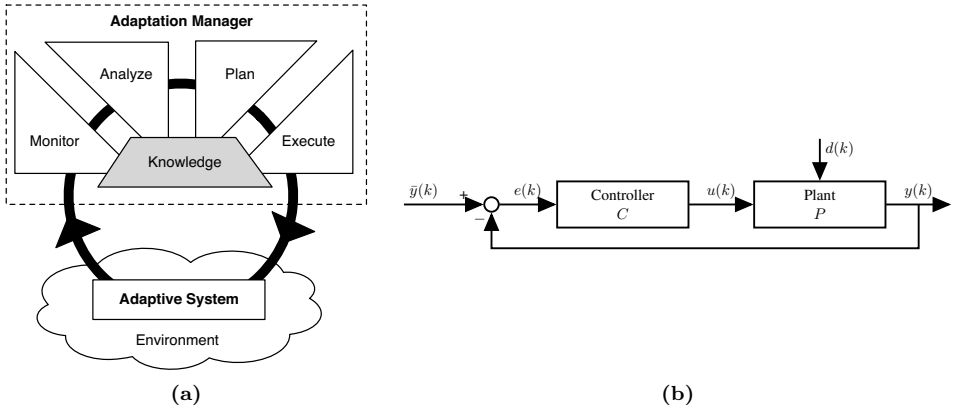


Figure 2.2: *The MAPE control loop (a) and an example of control loop from control theory (b). Figures reprinted from [142], Copyright © 2015, IEEE.*

2.4 Run-Time Supervision and Adaptation

Several approaches have been proposed in the literature for supporting the autonomic evolution of software. Numerous research areas have witnessed increasing interest in these issues. These include, among others, the Self-Adaptive Software, the Software Evolution, the Software and Requirements Engineering, the Autonomic Computing and the Multi-Agent Systems communities. The common goal is that of constructing software that monitors and modifies its own behavior in order to meet its high-level objectives despite possible changes in the operating environment. A self-adaptive system is intended as a system that is coupled with a so-called adaptation manager, which is in charge to modify the structure or the behavior of the system at run-time without interrupting its service [113].

Control theory and the MAPE feedback loop framework

Despite the many solutions proposed in the literature to make software systems self-adaptive, the vast majority of the frameworks share a *closed-loop* structure, where the software monitors requirements violations, plans counteractions if they are needed and enforces them [142].

Filieri *et al.* [142] identify a strict relationship between self-adaptive systems as intended from a software engineering point of view, and controlled systems, as per Control Theory, the study of mathematically grounded techniques for adaptation. In particular, the authors identify the MAPE (Monitor-Analyze-Plan-Execute) feedback loop [175], as the most popular framework in the literature for designing self-adaptive systems, and show the evident similarities between this framework and the control loop, which characterizes adaptive systems from a control perspective. Fig. 2.2, reprinted from [142], illustrates the two paradigms.

In the MAPE feedback loop in Fig. 2.2a, the *Adaptive System* operates in an

Environment which, as we have discussed earlier, can be dynamic and evolving on its own, thereby affecting the capabilities of the adaptive system to achieve its objectives. An *Adaptation Manager*, through its four main steps, monitors the system's behavior and collects data to detect changes (*Monitor*), analyzes data to assess the impact of changes and to determine if some reaction is required (*Analyze*) and, if needed, plans and executes actions in response to the changes (*Plan* and *Execute*). Similarly, a control loop like the one in Fig. 2.2b, mainly consists of a *Plant* and a *Controller*. The *Plant*, which can be associated to the *Adaptive System* of the MAPE feedback loop, determines an output $y(k)$ (indicating the value of the signal y generated at time instant k). The behavior of the *Plant* can be disturbed by *disturbances* $d(k)$, which may contribute to dis-align the behavior of the system from the desired behavior. The output $y(k)$ is compared with a desired output $\overline{y}(k)$ representing the objectives of the system, and an error $e(k)$ is determined, characterizing the distance between the actual and the desired behavior of the system. The error $e(k)$ is the input of the *Controller*, which has the task to determine new values $u(k)$ for configurable parameters, often called *knobs*, of the *Plant*.

It is not hard to see the parallelism between the two paradigms. Not surprisingly, Patikirikorala *et al.* [225], in line with Filieri *et al.* [142], identify an increasing trend in the usage of such type of solutions in self-adaptive systems.

Wang *et al.* [289], for example, propose a framework for the self-reparation of a system through reconfiguration. Their proposed architecture is composed by four components, which the authors associate to the main components of the MAPE feedback loop. The *Monitoring* component monitors the system's execution to detect possible failures, in terms of requirements satisfaction, and generates a log data. When failure occurs, the *Diagnosis* component tries to determine the causes of the failure, identifying the problematic entities of the system. The *Reconfiguration* component generates a number of possible system reconfigurations that are free of failures and select the best one among them, i.e., the one that contributes most positively to the NFRs of the system. Finally, the *Execution* component performs compensation actions to restore the system to a consistent state and reconfigures the system according to the selected reconfiguration.

Similarly, Dalpiaz *et al.* [100] introduce an architecture for adaptive Socio-Technical Systems that allows to switch between different requirements configurations at run-time when needed. The authors describe a Monitor-Diagnose-Reconcile-Compensate (MDRC) cycle, which: monitors and collects data about the behavior of actors in the STS and about the state of the environment; interprets data w.r.t. a given requirements model, diagnosing the causes of failure or under-performance if the requirements are not satisfied; searches for possible alternative configurations of requirements to deal with the problem; and performs compensating actions so to reconcile the actual and desired behavior of the system.

Almeida *et al.* [17] present a dynamic decision-making infrastructure that allows to monitor and reason about NFRs at run-time and to continuously select the features that compose a system configuration. They focus on Software Product Lines, which are represented by means of *feature models* [229] and permit to create a family of similar products (called product line) based on commonalities between members of the family. They extend feature models with properties that allow to specify, by means

of a language called DynamicNFR and inspired by RELAX, NFRs in such a way that they can be monitored at run-time. The architecture makes use of a monitoring system called *QoMonitor*, which retrieves data at run-time about the satisfaction of the NFRs. A *FMHandler* component analyzes the specification of the extended feature model given as input, and generates, by means of a genetic algorithm, the space of possible new system configurations. A decision algorithm, finally, selects a new configuration to adopt by maximizing an objective function generated from the NFRs.

Control theoretical concepts have been explored also in the MAS research community. Dastani *et al.* [110], for example, study normative multi-agent systems from a supervisory control theory perspective. In particular, they describe *norm enforcement* as a controllability problem. They interpret a multi-agent system as a *Plant* (as per above), whose behaviors are generated by a finite state automaton. The *Controller*, called *supervisor*, restricts the behavior of the multi-agent system by means of norms, which determine violating or compliant behaviors (words generated by the *Plant*). They distinguish controllable and uncontrollable events that can be generated by the *Plant* and in this setting, they describe the properties of three types of supervisors for a multi-agent system: regimented-based, sanction-based and repair-based supervisor. A regimented-based supervisor is a supervisor that prevents all controllable events that can make a behavior (sequence of events) norm-violating. A norm, then is regimentable if there are no *uncontrollable* events (i.e., events that cannot be controlled by the supervisor) that can make a behavior norm-violating. A sanction-based supervisor, instead, is a supervisor that allows violating behaviors to take place but imposes sanctions to punish violations. Imposing sanctions means to introduce additional *sanction events* to the sequence of events. To do so, they employ a *virtual Plant*, so to avoid the possibility of causing violations by means of introducing events in the original (not virtual) sequence of events. Similarly to the case of regimentation, a norm is sanctionable if there are no uncrontrolled events that can make the behavior norm-violating. Analogous concepts are presented also for reparation-based supervisor, where they consider so-called *repairing events*, that can follow a potentially violating behavior to make it non-violating.

In this dissertation, we follow the MAPE paradigm. In the examples of frameworks reported above, the focus is on adapting the configuration of the system in order to restore, as much as possible, its compliance with the requirements defined at design-time. Differently, in this dissertation we tackle settings in which this is not possible, or it would be more expensive than *changing the requirements* themselves. The supervision of the system that we consider is at the requirements level. In our framework, outlined in Fig. 1.3 and Fig. 1.4, we consider as goals of the feedback loop that determines if and how the behavior exhibited by the system deviates from the desired one, an explicit representation of the system's objectives, which we will sometimes call *system-level objectives*, or *system's objectives*. The *Plant*, or *Adaptive System*, under control corresponds to the target *System*, which is a normative multi-agent system whose behavior is constrained by a set of requirements (norms). The *Monitoring* component monitors the violations of the requirements, but also the achievement of the system's objectives, distinguishing data in different operating

contexts. The *Supervisor* corresponds to the *Controller*, or *Adaptation Manager*. Its *Assumptions Validation* component in the *Revision Engine*, similarly to the *Analyze* component of the MAPE control loop, performs a diagnosis of the norms when the objectives are not (fully) achieved, i.e., it detects if some of the *design-time assumptions* related to the requirements are, or became, invalid. For example, the component determines whether the requirements are satisfied in certain contexts, or if their satisfaction is beneficial or harmful for the system's objectives, and what is the most likely explanation for the system's objectives not being achieved. The *Revision Policy Application* and *Revision Operations* components then, similarly to the *Plan* and *Execute* components of the MAPE control loop, try to determine *new requirements* for the system, by first suggesting types of revisions needed (for example whether a requirement should be weakened, or made more strict) and then by finding new requirements that fit the suggestions as much as possible. In a first step, in Chapter 3, we focus on the validation of the design-time assumptions and propose novel algorithms to suggest different types of revisions for the current requirements based on the run-time validity of the assumptions. Given the suggested revision we search a space of possible predefined alternative configurations of requirements to find the ones that are most aligned with the suggestions. Later, in Chapters 4 and 5, we will make a step forward and we will study how to synthesize new requirements based on the given suggestion.

In the following, we provide some further details on the state-of-the art concerning the different steps performed in the MAPE loop.

2.4.1 Monitor

In formal settings, the typical approach to check at run-time if a system satisfies certain properties is *runtime verification* [35]. Runtime verification typically consists of automatically translating a property, expressed in a formal language, into a so-called *monitor*. The monitor is then deployed and used at run-time to verify if the execution of the system, typically represented as one or more *traces* (sequences of the states of the system), complies with the property being verified.

The type of monitor that is synthesized for a certain property strictly depends on the language in which the property is expressed. Several works have studied properties expressed in Linear-time Temporal Logic (LTL) or some of its extensions such as past time LTL, or Timed LTL [35, 161, 307]. Some monitors employ formula rewriting or formula progression techniques [28], a process that, without going into the details here, iteratively consumes states from a trace and generates a new formula from the original one that must be satisfied by the following states. If a state that does not satisfy the formula before it is completely consumed is encountered, a violation of the property is detected. Other approaches translate the original property into efficient dynamic programming algorithms (the monitors), while others translate the property into finite-state automata, or construct so-called testers [228].

The main advantage of runtime verification is that monitors are typically tractable and allow to verify properties in linear or polynomial time w.r.t. to the size of of the original formula [161]. Runtime verification, therefore, enables the runtime monitoring of properties and mitigates the issue of an exhaustive verification at design-time,

intractable for complex systems. Furthermore, runtime verification provides good-enough solutions for dynamic systems where the context changes at run-time and unforeseen situations may happen, as it does not require a complete representation of all aspects of the system.

To briefly give a more concrete idea of a simple monitor, we report in Algorithm 1 the monitor to detect violations of the conditional prohibitions described in Sec. 2.3. The algorithm, taken from Alechina *et al.* [10], returns *true* if a conditional prohibition n is violated in a state $t[i]$ at position i on the trace t, and *false* otherwise. Note how the state of the norm (in particular the fact that the norm is detached) is maintained while the trace is sequentially analyzed. If a state where the prohibited state is encountered after the norm has been detached and before a state where the deadline holds, then a violation is detected. A similar monitor for conditional obligations can be found in [10].

Algorithm 1 Violation of a Conditional Prohibition [10]

Input: finite trace t; norm $n = (c, P(\phi), d)$
Output: *true* if t violates n, *false* otherwise

```
detached ← false
for i ∈ [1, t.length] do
    if t[i] ⊨ c then
        detached ← true
    if detached then
        if t[i] ⊨ φ ∧ t[i] ⊭ d then
            return true
        else if t[i] ⊨ d then
            detached ← false
return false
```

Among others, in the context of normative agents, a study of how to specify norm monitors is provided by Bulling *et al.* [69]. They provide a logical and computational framework to specify various types of norm monitors. They represent norms in LTL and analyze a number of LTL-based monitors, formally studying the relations between monitors and norms and the computational complexity of the monitors.

Requirements monitoring frameworks

Numerous requirements monitoring frameworks can be found in the literature as well. Two seminal work about requirement monitoring in relation to assumptions validation are the ones of Fickas and Cohen. Fickas *et al.* [140] describe a process where they subdivide top-level soft requirements into pieces. This enables the identification of the assumptions underlying each of those pieces and to associate remedial actions to apply when the assumptions are violated. This process is shown as crucial in order to determine what to monitor. Monitoring the assumptions allows them to trigger the predefined adaptation that can be done by the system itself or suggested to the designer. Cohen *et al.* [88], instead, describe a prototype monitoring system called AMOS (Assumption Monitoring System), where a compiler automatically converts

expressions written in a language called FLEA (Formal Language for Expressing Assumptions), into runtime monitoring code that observes the behavior of the system and of the environment w.r.t. the expressed assumptions.

More recently, Robinson *et al.* [236], presented ReqMon, a requirements monitoring framework for enterprise software. The framework allows to monitor requirements expressed in KAOS, extended with aggregate functions that permit for example to evaluate temporal requirements. In ReqMon, KAOS requirements are mapped to observable events. Requirements are then evaluated incrementally as the events arrive, by using four possible states of an evaluation of a property: undefined, satisfied, violated and partial.

Wang *at el.* [290] present a framework that allows to monitor (and diagnose) requirements expressed by means of goal models. They annotate a goal model by associating each goal/task with preconditions and effects and a switch that enables or disables the monitoring. At design-time the software is instrumented according to the annotated goal model so to enable the runtime monitoring. At run-time, the program generates log data, which is collected and can then be analyzed and diagnosed. The actual analysis is performed offline. The annotated goal model is translated into a propositional logic formula which is then fed, together with the log data, to a SAT solver that determines if the traces in the logged data do not satisfy the requirements.

Filieri *et al.* [143] describe the WM framework for efficiently conducting probabilistic model checking at run-time. While traditional model checking approaches are typically not applicable at run-time, here the authors propose a precomputation step that simplifies the work that needs to be done at run-time. They focus on NFRs such as reliability or performance requirements, expressed as PTL (in particular PCTL), which they check against a markov model (in particular DTMC). The design-time precomputation determines a set of symbolic expressions that depend on variables whose values can be binded only at run-time. They propose two possible approaches to compute such expressions: a matrix-based approach, useful for large systems with few parameters, and an equation-based aproach, useful for cases with many parameters and for non-parallel problems. At run-time then the system has only to bind the value of such variables and verify their validity. In order to make this possible, a number of assumptions are made, such as the possibility to anticipate potential changes so to restrict them to a subset of environment parameters that can modeled as variables.

Runtime monitoring in this dissertation

In this thesis, we use runtime monitoring to collect data about the behavior of the system in terms of satisfaction of requirements (or violation of norms) and achievement of system's objectives in different operating contexts. We distinguish between three main elements that we will monitor.

- *Context.* We will assume that a number of contextual properties concerning measurable environmental factors are determined to be monitored. Examples are the *time of the day*, or the *type of weather*. These properties will have different possible values according to their domain, for example *day* or *night*,

or *rain* or *sun*. We will call *operating context* a combination of values of these properties.

- *Requirements* (or *norms*). We consider computationally verifiable requirements and norms, as per Sec. 2.2, and we will monitor their satisfaction or violation. For example we will consider norms regulating the *maximum speed* or the *minimum safety distance* that vehicles should maintain when driving on a smart road. Vehicles will autonomously decide whether to comply or violate with such norms, taking into account the consequences of a violation. In most of the cases we will assume that perfect monitoring is available and that any violation of the norms can be detected when it occurs. In some of our experiments, however, we will also consider imperfect monitors which can misdetect some violations.

- *System's objectives*. We will consider an explicit representation of the objectives of the system that are desired to be achieved by means of enforcing norms. For example a possible objective may concern *user satisfaction* about the system, or *no traffic jams in the city*. We distinguish these from requirements as we associate them to the raison d'être for the requirements [303]. Furthermore, even though for the sake of our simulations we will consider computationally verifiable objectives, we support also non-computationally verifiable ones (for instance the user satisfaction cannot be verified computationally, unless humans provide an input). To give a practical example, if we are considering an airport software system, and the objectives of the stakeholders include the satisfaction of the customers, we envision that an evaluation of customer satisfaction could be obtained by means of the notorious *HappyOrNot* feedback terminals that can be positioned in the airport for measuring customer satisfaction.

One the one hand, therefore, we employ runtime monitoring to *measure* the operating context and the achievement system's objectives. On the other hand, we employ it to *verify* the requirements. We use then the collected run-time data to analyze and diagnose the requirements and to guide their revision when needed.

2.4.2 Analyze and Diagnose

Once run-time data is collected through monitoring, it needs to be analyzed to determine if some adaptation is required and possibly to understand what are the root causes of the detected problems.

Wang *at el.* [290], in the framework earlier illustrated, perform an offline SAT-based diagnosis of the annotated goal model. This is made possible by a translation of the goal model into propositional logic so that a series of axioms of deniability of goals, tasks, decompositions and contribution links of the goal model, i.e., axioms that describe when these elements are made invalid, can be checked with data collected at run-time.

Viana *et al.* [285] present a prototypical framework called MaCoRe_SoS (Managing Conflicting Requirements in Systems of Systems): following the MAPE paradigm, MaCoRe_SoS aims at identifying, diagnosing and resolving conflicts between requirements in a SoS. To express the requirements (which are maintained at run-time) they

use an extension of the RELAX language in which they introduce concepts concerning resources affected by the requirements and events that can violate the requirements. During the diagnosis phase, they focus on the identification of conflicts. In particular, when an event notification is received at run-time, the requirements related to the event are checked in order to detect the affected resources and, if some of the assertions generated at design-time starting from the original requirements is violated, a conflict is reported.

Work related to the analysis of requirement assumptions is mainly proposed by Ali *et al.* [15, 16]. In [15], the authors describe a mechanism for analyzing the validity of design-time assumptions in order to support autonomic evolution of requirements. In particular, data collected at run-time is used to determine which of two possible software variants, assumed at design-time to be equally able to meet a certain requirement, succeeds more often in its intent. In [16], the authors develop a set of automated analysis mechanisms to support the requirements engineers in detecting and analyzing possible modelling errors in contextual requirements models. They propose two automated reasoning mechanisms to detect two kind of modelling errors: inconsistent specification of contexts in a goal model, and conflicts between tasks.

Bayesian Networks

Tools for diagnosis and for reasoning under uncertainty are many, and variegate. The literature spaces from Fault Tree Analysis [132], to logical approaches, via rule-based ones, via solutions that consider ignorance as opposed to uncertainty for performing inference, such as the Dempster–Shafer theory [250], via fuzzy approaches that allow for more "vague" inferences than more classical probabilistic approaches, by considering that a proposition can be "sort of" true [240]. One particular tool that has been widely used to perform diagnosis and for the intelligent analysis of probabilistic distributions is Bayesian Networks. Bayesian Networks have been extensively used in many fields (ranging from medicine to forensics) as knowledge representation structures for learning and reasoning about the inter-dependencies between their variables [240].

In this thesis we also choose to use Bayesian Networks for a number of reasons. They are a well established and powerful models, which allow for expressing any possible full joint probability distribution. They are general enough to be used for probabilistic reasoning both in static contexts, and in dynamic ones, for example via Dynamic Bayesian Networks, which encompass other dynamic models such as Hidden Markov models and Kalman filters [240]. While other powerful tools for reasoning about causality have been presented in the literature, e.g., models of concurrent non-deterministic computations, such as Petri Nets, or Causal Nets [227, 273], these are often used to analyze or discover processes at a more fine-grained level of detail, rather than at the requirements-level, which is the scope of this thesis. Nonetheless, several works have shown the several similarities and analogies between these causal models and Bayesian Networks, also proposing mappings between the two structures [171, 187]. Moreover, given their simplicity, flexibility, and popularity, tools and libraries supporting Bayesian Networks are widely available and well supported (e.g., among others, the bnlearn R library, or the bayespy Python library), alongside with graphical tools to visually explore and interact with the network such as the Netica tool [1].

Finally, thanks to the possibility of handling missing data, to incorporate expert knowledge with historical data, and to their intuitive and graphical representation, in the Software Engineering literature Bayesian Networks have already been adopted as a support tool for diagnosis and decision making. Such models have been used, for example, to evaluate software reliability [127], to estimate software effort [204], for software maintenance [114] or for defect prediction [138]. Furthermore, they have been employed also for the runtime verification and diagnosis of requirements and to perform or support autonomic decision making at run-time [141, 226].

Despite their wide adoption, Bayesian Networks haven't been used to reason about validity of assumptions expressed in requirements models. In this dissertation, we propose to use Bayesian Networks as a run-time counterpart to requirements model and we use them to collect statistical information about the behavior of the system (in terms of the three main elements described above), to automatically evaluate a wide range of design-time assumptions that are made by the designer of the system, and to inform and guide automated strategies to revise the current requirements. For this reason, we provide here some background about Bayesian Networks. The comparison of our approach that employs Bayesian Networks with other approaches is left for future work.

A Bayesian Network [240] is a data structure that graphically and quantitatively represents the (probabilistic) dependencies among variables. In particular a Bayesian Network (Fig. 2.3 provides an example) is a directed acyclic graph (DAG) where:

- Each node represents a random variable in probability theory, each of them with a domain (i.e., the set of possible values it can take). For example, the domain of a random variable *Weather* could be $\{sun, rain, cloudy\}$, or the domain of a random variable *SpeedLim*, representing a requirement, could be $\{sat, viol\}$.

- A set of directed arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is called *parent* of Y. In a Bayesian Network there are no cycles. The nodes and arrows of a Bayesian Network, together, are called the *structure* of the Bayesian Network.

- Each node X is annotated with a conditional probability table (CPT). Each row in a CPT contains the conditional probability of each node value for a so-called conditioning case (i.e., a possible combination of values for the parent nodes). A conditional probability, sometimes called *posterior* probability, is the probability that an event happens (i.e., that a random variable assumes a given value in its domain) given some other revealed piece of information, called *evidence* (it is given the value for some other random variable). For example $P(SpeedLim_{sat} \mid Weather_{sun})$ indicates the (conditional) probability that requirement *SpeedLim* is satisfied given that the weather is sunny. Such probability is 0.8 in Fig. 2.3, and is indicated in the top left cell of the CPT of node *SpeedLim*.

  Conditional probability is distinguished from unconditional, or *prior*, probabilities as the latter refer to the degree of belief in a certain event in absence of any other information. For example $P(SpeedLim_{sat})$ indicates the (prior)

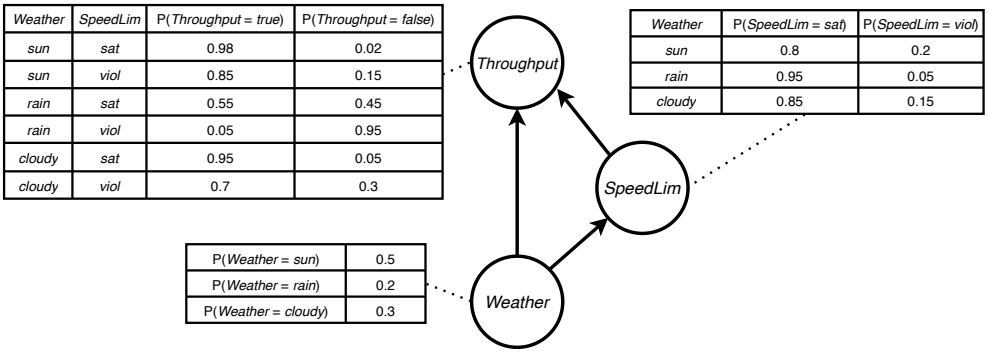


Figure 2.3: *An example of Bayesian Network.*

probability that *SpeedLim* is satisfied, regardless of the weather conditions, or $P(Weather_{sun})$ indicates the (prior) probability that the weather is sunny.

Each row of the CPT sums to 1, as the entries represent the exhaustive set of cases for the values of variable, given a combination of values of the parent variables. The entire CPT of a node X represents the conditional probability *distribution* $P(X \mid Parents(X))$, which quantifies the effect of the parents on the node.

The conditional probability values are the *parameters* of the network. These parameters can be provided manually by domain experts, learned from data, or a combination of both.

2.4.3 Plan and Execute

The knowledge acquired and diagnosed at run-time can be leveraged to trigger and perform an adaptation of the system. Most of the existing frameworks focus on automatically restoring the compliance of the system with the already defined requirements by searching for a new system configuration that is expected to outperform the current one in achieving the requirements, or on selecting more appropriate requirements variants [32, 185].

Non-functional requirements (NFRs) have been mainly used to trigger and guide self-adaptation. For example, Salehie *et al.* [241] describe an automated decision-making mechanism for run-time selection of adaptation actions. NFRs, which are used as the objectives of the adaptation, drive the selection. The adaptation actions selection mechanism that they propose is modeled as a game between goals, in which each goal aims at selecting its preferred adaptation action (a preference list is given for each goal). A voting mechanism is then employed to weight the votes from the goals that are activated (i.e., denied) at run-time.

In the reconfiguration components of their framework, Dalpiaz *et al.* [100] search for the alternative system configurations that maximize the satisfaction of the NFR, based on the contribution links expressed in a goal model.

Bencomo *et al.* [42] propose the use of Decision Networks (an extension of Bayesian Networks including decision nodes) to automatically choose between different system's configurations, based on NFRs. They model uncertainty associated with the satisficement of NFRs by using probability distributions conditioned by the possible configurations of the system. Decision nodes in the network are associated to the choice between different system configurations. The expected utility of taking a decision is computed by making use of weights associated to NFRs. Evidence in the network is associated with claims which represent assumptions made at design-time, whose validity is monitored at run-time, and with environmental properties representing uncertainty factors (e.g. a failure of a system component).

In a similar fashion, Paucar *et al.* [226] propose techniques to reassess at run-time the assumptions made at design-time about the weights of NFRs. Given data collected at run-time, they employ Dynamic Decision Networks to revise the weights of the NFRs.

Serral *et al.* [249], propose run-time algorithms for the selection of optimal tasks when adaptation is triggered by changes in user preferences, by faults in the execution of some tasks, by plan failures or by context evolution. In order to select the best set of subtasks, they evaluate the contribution of the possibilities to the NFRs.

Some, even though not many, approaches, deal with the adaptation of the system at the requirements-level, i.e., they tackle the problem of revising the requirements of the software, instead of adapting the software to restore its compliance with the requirements. As adaptation at the requirements-level is also the main scope of this thesis, we overview requirements (and norms) revision more in detail in the next section.

2.5 From Model-Driven to Data-Driven Revision

Historically, a formal account to the problem of revising a system, its requirements, and its norms, has roots in the context of theory and belief change, that is the study of how to *correctly* change a representation of knowledge. Alchourron, Gardenfors, and Makinson [6, 148], in their seminal work on the subject, define a framework which is now known as AGM, from their names. They describe the main aspects of theory change, and in particular three principal forms of change: *expansion*, i.e., the introduction in the current theory of a new formula that is consistent with the theory; *contraction*, i.e., the removal of a formula from the current theory; and *revision*, i.e., the introduction in the current theory of a new formula that is not consistent with the theory. They illustrate that, while expansion operations are typically not problematic, contraction and revision operations are less trivial. This is because when introducing a new formula that is inconsistent with the current theory (or removing an existing one), in order to obtain a new theory that is consistent, we must also remove some of the existing parts of the theory. The main question becomes then how to define the operations of contraction and revision so that appropriate formulae are chosen to be removed from the current theory.

The AGM framework inspired, more or less explicitly, a number of works on the

revision of rules, requirements, and norms aimed at governing software. In particular it influenced a number of *model-driven* revision approaches, which rely on a model or on a theory of the system and of its operating environment to formally characterize and reason about change. While the focus of this thesis is on *data-driven* revision of norms and requirements, and we do not assume that a model or a theory of the system is explicitly available, we briefly report here also on some model-driven revision approaches, so to provide a comprehensive review of the literature.

2.5.1 Model-Driven Revision

MacNish *et al.* [199] were among the first to describe how theory change could be usefully applied to revising requirements, modeled by means of *hierarchical goal structures*. In about the same years, also Zowghi and Offen [310] presented a logical framework where they used the theory of belief revision to model and reason about the evolution of requirements. They argue that software engineering is concerned with building and managing theories, and that requirements evolution consists of mapping one theory to another.

In the context of normative systems and artificial societies, Shoham *et al.* [254], in presenting results about the problem of synthesising social laws that guarantee the achievement of system objectives, introduce the idea of *more restrictive* social laws. A social law (a set of constraints that prohibit agents to perform actions in states that satisfy certain conditions) sl_1 is more restrictive than a social law sl_2 if for every constraint in sl_2 there is a constraint in sl_1 that prohibits the same action but in states that satisfy less specific conditions. While the idea does not directly refer to revision operations, it describes a *relationship* between two social laws, one more restrictive than the other, which relates to the ideas of contracting or expanding a law. More explicitly related to norm revision, Boella *et al.* [53] present a framework for normative change. Taking input/output logic [201] as a starting point, they define norms as pairs of (propositional) formulae (a, x), to be read *if a then x is obliged*, and they map the AGM operations to corresponding operations on norms.

In the context of requirements analysis, Lamsweerde *et al.* [276] propose a number of techniques to resolve conflicts and divergences from the specification of requirements, represented as goals within the KAOS framework. Divergences (logical inconsistencies) are resolved by introducing new goals or by *transforming* the specifications of the goals towards a conflict-free one. Logical inconsistencies between goals often derive from so-called boundary conditions: situations where particular circumstances make goals conflicting. In order to avoid or deal with boundary conditions, Lamsweerde *et al* describe formal strategies such as the following.

- *Avoiding Boundary Conditions*: logically derive, from the available domain theory and from the logical model used to represent the system, a new goal that will prevent the boundary condition to happen.

- *Goal Restoration*: when boundary conditions cannot be avoided (e.g., when they involve entities out of the control of the system), introduce a new goal that states that if the boundary condition occurs then the goal will become

true again in the (reasonably near) future. In other words, goal restoration temporary disables a goal.

- *Goal Weakening*: make a goal more liberal so that the boundary conditions are accepted. This can be done by using syntactic generalization operators, such as adding a disjunct, removing a conjunct, or adding a conjunct in the antecedent of an implication, in the same fashion of theory revison.

- *Alternative Goal Refinement*: obtain alternative sub-goals which are not divergent, or whose divergence can be resolved by using some strategy.

Jiang *et al.* [167], instead, discuss the contextualization of norms. They study how to refine norms to make them suitable for specific contexts. They explicitly represent the context of application of a norm and they introduce contextual refinement normative structures, which are used to organize and refine the norms. Norms are expressed according to the ADICO syntax [222] (e.g., for regulative norms: *[ATTRIBUTES] [DEONTIC] [AIM] [CONDITIONS] [OR ELSE]*). Contextualization is described as a relationship between two norm sets such that the new set elaborates the old with refined normative components; the new set adds new norms; the new set removes some of the norms; or the new set elaborates the interrelations between norms. Each norm set, then, can be associated to a certain context. The authors also present a mechanism that translates the hierarchy of norms into Colored Petri Nets, so that properties such as reachability and liveness can be verified.

The operations proposed by Lamsweerde, Jiang, *et al.* inspire our work. In their work, such strategies are applied at design-time, they rely on the available domain knowledge which is characterized in a formal model, and they are employed to perform conflict resolution or to guarantee formal reachability and liveness properties of the system. In our framework, we will adopt somewhat similar strategies but at runtime, based on statistical execution data, and with the explicit goal of improving the alignment of the requirements with the explicitly represented system's objectives.

Alechina *et al.* [9] describe an operation called *normative update* as the operation of introducing a norm set in a MAS represented as a transition system. In particular, the update is performed by modifying the transition system. A regimented norm, for example, is introduced by removing paths or transitions from the system. A sanctioned norm, instead, is introduced adding a proposition *san* in the states reached by violating the norm.

Similarly, Knobbout *et al.* [179] describe how to update a normative system when adding or removing action-based norms called *to-do* norms. These are norms that ensure that after the execution of a certain prohibited action in specified states, some normative facts indicating the violation become true until a repair action is performed. The authors represent the normative system as a pointed labelled transition system and the normative update is an operation on such system, such as duplicating states and adding or removing propositional symbols indicating the violations in the opportune states, similarly to [9]. In a follow-up work [180], the authors extend the framework to support also state-based norms called *to-be* norms. In addition, they define a dynamic logic with a norm update operator that describes the operation of adding a norm.

Alechina *et al.* [10] take also another point of view to norm revision and they introduce the concept of norms approximation with respect to a monitor. An approximated norm is synthesized from an original norm to maximize the number of violations that an imperfect monitor can detect.

All the approaches mentioned above, describe techniques that rely on a model or a theory of the system. As we have discussed, however, in complex modern systems, it is often the case that the available information is only partial or imprecise. In highly dynamic environments, furthermore, the available model may become outdated very quickly once the system is deployed, and these techniques are not sufficient anymore to deal with the continuous evolution of software systems. For this reason, more data-driven approaches have started appearing for the run-time revision of norms and requirements, also allowing their integration within self-adaptation frameworks. This is also the scope of this thesis. We overview here a number of approaches specifically focused on the revision of norms and requirements driven by data.

Since the two communities of requirements engineering and normative multi-agent systems, despite their similarities and commonalities, have been conducting work in a relatively separate fashion, we distinguish, for simplicity, approaches to the revision of requirements, and approaches to the revision of norms. Some approaches, however, such as the ones using Inductive Logic Programming (ILP) [129], or work on the contextualization of properties, have been studied in both communities.

2.5.2 Data-Driven Requirements Revision

Several existing approaches to data-driven revision of requirements in the literature mainly focus on non-functional requirements [17, 41].

Bencomo *et al.* [41], for example, employ Dynamic Decision Networks (DDNs), an extension of Dynamic Bayesian Network to support decision-making, to suggest a revision of the *priorities* associated to non-functional requirements, based on a degree of uncertainty of events in the environment. They model the uncertainty about the satisfaction of NFRs by means of probability distributions conditioned by a given configuration of the system (pre-established design alternatives). In the DDN, decision nodes are meant to determine one of the possible configurations of the system and the utility of each configuration is the expected utility of assigning certain weights (the priorities) to the NFRs. They also provide evidence (run-time data) to the network in the form of claims about the current run-time validity of assumptions made at design-time. This allows the DDNs to be employed as a run-time tool to continuously adjust the priorities given to the NFRs, when the validity of the assumptions change.

In a later work [39], the authors introduce the concept of surprise–a considerable divergence between the belief distribution in the DDN prior and posterior to the occurrence of an event–as a measure for the quantitative analysis of uncertainty and deviation of a system from its normal behavior. They argue that small surprise could be used as a suggestion that a set of NFR can be RELAXed temporarily (i.e., in terms of revision, replaced with another pre-defined RELAXed NFR) in order to tolerate unanticipated, but transient environmental conditions. This could potentially avoid unnecessary further more costly adaptations of the system.

Building up on Bencomo *et al.*'s work, Ramirez *et al.* [232] propose AutoRELAX, an approach to RELAX goal models to address adverse environmental conditions. In particular they consider a KAOS goal model of the functional requirements of the system, where each goal is designated as either invariant (cannot be revised) or non-invariant. Starting from this model, they generate a solution space comprising all possible RELAXed goal models and they explore it by using a genetic algorithm (a stochastic search-based heuristic for efficiently solving optimization problems, where a fitness function is used to evaluate the quality of each possible solution and guide the search process towards promising areas–i.e., areas where the solutions are expected to have high quality–in the solution space). The requirements engineer associates to each non-invariant goal, which is mapped to a gene for the genetic algorithm, a utility function that can represent the degree of satisfaction of the goal. The utility functions are then used by AutoRELAX to evaluate, during a number of simulations performed by the genetic algorithm while exploring the space, how goal RELAXations will affect the behavior of the system in response to possible sources of uncertainty. Such possible sources of uncertainty are defined by the engineer in terms of their likelihood and causal relation with the goals, to which the system may be exposed. In AutoRELAX, therefore, data is obtained by means of simulations and requirements revision is interpreted as goal model revision, and in particular goal model RELAXation, which practically means replacement of the current goal model with a new RELAXed one.

In this dissertation, and in particular in Chapter 3, we adopt an analogous strategy to Ramirez *et al.* Instead of using genetic algorithms, however, we will propose the use of hill-climbing optimization techniques to quickly search the space. This provides a strategy to explore the space of possible system's configurations that can be directly applied at run-time without relying on simulation, even though supporting it if desired. Furthermore, instead of using a fitness function defined by the system's designer for evaluating the quality of possible solutions, we adopt a probabilistic approach where the probability distributions are automatically learned at run-time based on execution data. Finally, in Chapters 4 and 5, we will make an additional step forward and we will study some strategies to *synthesize* new requirements at run-time based on data, without relying on a predefined fully specified set of alternatives.

Knauss *et al.* [177] discuss the use of data-mining to deal with run-time uncertainty and to determine adequate contexts where to apply requirements. They present an approach, called ACon (Adaptation of Contextual requirements), that uses a feedback loop that maintains contextual requirements up-to-date based on run-time data. Practically, their approach (re-)operationalizes the applicability contexts of contextual requirements based on run-time data. This approach is similar to the one we adopt in this dissertation. We also use data to select and determine appropriate requirements for each of them. In addition to the requirements contextualization, in this dissertation we will also focus on validating the assumptions in different operating contexts, and on revising not only the applicability context of the requirements but also their content.

Recently, Bennaceur *et al.* [44] proposed a resource-driven requirements adaptation approach. They describe a framework where three steps of adaptation are performed if some of the current requirements are not satisfied. In the first step, the requirements are not revised, but the system tries to find, by means of a multi-objective

constrained optimization problem, a subset of the available resources that will satisfy the requirements. If that is not possible, the system attempts to replace some of the resources with alternative ones that are semantically equivalent. This is done by using a similarity relationship between the different resources. If no replacement is found, finally, the system adapts the requirements by selecting a subset of them that can be satisfied using the available resources. Data, therefore, consists of the run-time available resources, and the adaptation of the requirements amounts to finding an alternative subset of requirements from the set of all possible requirements that can be satisfied by the available resources.

This approach is somewhat analogous to the approach of Ramirez *et al.* [232], as they also search for an alternative *subset* of predefined requirements. Both approaches mainly focus on *relaxing* (or weakening) the requirements when it is not possible to satisfy them. In this dissertation, we will not limit ourselves to relaxation of predefined requirements, but we will propose mechanisms that support their general revision, detecting when it is necessary to relax them, but also when it is necessary to *strengthen* them or to *synthesize new ones*. Furthermore, we will also consider the revision of *sanctions* used to enforce the requirements. This is an alternative strategy to revise requirements when they cannot be satisfied, especially suitable for weakly controllable systems like Socio-Technical Systems: if agents do not obey our requirements, we can increase the sanctions to motivate them to obey.

Closer to the concepts of agents, Dalpiaz *et al.* [98] describe a number of strategies that agents in open systems can employ to adapt their behavior in response to some external trigger. They focus on open systems characterized by the interactions, modeled in terms of *social commitments*, among autonomous and heterogeneous agents. Agents' goals are represented by means of hierarchical goal models. These models, since characterized by AND-OR decompositions, determine a number of possible alternative variants of the model (roughly a subset of goals in a goal model) that can be selected in different situations in order to satisfy a target goal. The authors describe adaptation operations such as: choosing a different variant to satisfy a target goal, if the agent believes that the current strategy will not succeed because the fulfilment of a commitment made by another agent, necessary to support the current strategy, is threatened; selecting a different variant that includes redundant ways to satisfy a goal to reduce the risk of not achieving critical goals; taking more commitments in case of low trust in the other agents; etc. Their work focuses on the goals of individual agents, and not on the goals of an *institution*. The adaptation operations, practically consist of switching to alternative commitments, or selecting an alternative variant of the goal model when the current one appears to the agent to be risky or threatened (these run-time beliefs of agents can be seen as the run-time data).

2.5.3 Data-Driven Norm Revision

Agent-focused revision approaches

In the context of MASs, run-time data-driven approaches to the revision of norms have mostly focused on social norms and took the point of view of individual agents. This is due to a shift of interest in the recent years from a legal to an interactionist point of view on normative multi-agent systems. This shift was also observed by Boella

et al. [52]. In this sense, norms are typically considered as patterns of behaviors that emerge within a population of agents. A norm is assumed to have emerged if a significant part of the population adheres to a certain behavior, without an explicit external authority enforcing such behavior. In order for norms to emerge, agents are endowed with capabilities that allow them to sense, recognise and reason about the behavior of other agents (data from the point of view of the agents) and to react to it, for example imitating the behavior and adjusting to the observation their beliefs about the norms holding in the system., e.g., by replacing the current beliefs of norms being in place with new ones.

Norm revision in this context is related to *norm emergence*, *learning* or *identification* by the agents in a population. Some approaches to norm emergence use ideas from evolutionary game theory to propagate over time norms through a society [26, 213]. In these approaches, norm revision can be seen as the replacement of a norm with another from the point of view of the agents, and it is based on the norms that the majority of the agents in the society selects (the data). Other approaches endow agents with learning capabilities that allow them to learn which (social) norms, or conventions, are present in a system or are better to achieve their goals [5, 254]. Cranefield *et al.* [95], for instance, present a Bayesian approach to norm identification. They show that agents can internalize norms that are already present in an environment, by learning from both norm compliant and norm violating behaviors. Agents have knowledge (in the form of a directed graph) about all possible actions other agents can perform in certain states, and norms are state-based norms extended with temporal operators. Mahmoud *et al.* [200] focus on agents joining an open MAS who have to learn the unstated norms and propose an algorithm for mining regulative norms that identifies recommendations, obligations, and prohibitions (in the form of action-based norms) by analyzing events, corresponding to actions of agents, that trigger rewards and penalties. In these approaches, data provided to the agents corresponds to execution traces representing the behavior exhibited by other agents, which is labeled as either obeying or violating by means of sanctions. The revision of the norms, even though it is not explicit, can be seen as the *process* of identifying and learning the norms that are more likely to exist in the system, given the data. In a similar way, several works have been presented in the context of *negotiation* and argumentation, where agents are enriched with negotiation capabilities and through a negotiation process agree on which norms to adhere to at run-time [23, 51].

Institution-focused revision approaches

All the agent-focused approaches mentioned above require, or rely on, a number of capabilities of the agents participating in the system and focus on social norms. As we have seen, however, one of the main characteristics of modern software systems is that its components are heterogeneous, as they represent (and can be implemented by) different parties, and their internals, in line with the fundamental principle of neutrality of open systems, may be only partly known to the system designer. A small part of the literature also explored data-driven approaches concerning regulative norms, instead of social norms. This is also the scope of this dissertation.

Bou *et al.* [59], for example, address a problem related to ours, where they intend

to devise an Autonomic Electronic Institution (analogous to an adaptive normative MAS) that is able to revise the norms that regulate agents behavior in order to achieve some overall objective. They use expressions of the form *A causes F if* $P_1, \ldots, P_n$ to define regulative prohibitions. In order to revise the norms, they associate to each of them numerical parameters, which can be related to *sanctions*, and they adapt these parameters by means of genetic algorithms. Starting with an initial set of norms and applying evolutionary learning, they learn a function, called normative transition function, that determines optimal parameters of the norms that maximize an objective function characterizing the objectives of the system (e.g., minimize the number of accidents in a city). This learning step (optimization of the sanctions) is performed at design-time, and the authors argue that at run-time it is then possible to use case based reasoning to choose (the revision) which norm set is better for the actual population of agents (the run-time data) by checking the similarity with the populations used during the learning process at design-time. In a traffic regulation experiment, they learn the optimal sanctions and number of necessary police agents for enforcing a predefined *right-priority* norm with different populations.

Bou *et al.*'s research is mostly related to our proposal to sanctions revision discussed in Chapter 4. Unlike them, however, we apply our learning and adaptation mechanism directly at run-time. While to help directing the revision we make use of some knowledge that the system designer can have, or obtain from data, about agents' preferences, we do not rely on design-time simulation, which requires to model the dynamics of the system.

In a similar fashion, Cardoso *et al.* [72] present an approach to adapt at run-time deterrence sanctions (fines) in a normative MAS. They obtain run-time data by measuring how often an obligation is used and how often it is violated, and they determine the strength of a fine as directly proportional to its application frequency. In particular, they define a threshold for the tolerated number of violations. Fines of obligations are continuously changed at run-time so that if the number of violations goes below the threshold, the sanctions are decreased, otherwise they are increased. Differently from this approach, our revision of the sanctions is driven by the achievement of the system's objectives. This is a crucial difference as it relaxes the assumptions made by Cardoso *et al.* that the enforced norms are *effective* for the achievement of the system's objectives. In our case, we increase sanctions only if it is confirmed at run-time that it is important for the system's objectives that the norms are obeyed (and they are not already sufficiently obeyed), while we decrease them if it is not necessary that all agents obey the norms to achieve the system's objectives.

Miralles *et al.* [208] study how an organisation can adapt MAS regulations at run-time. They present an abstract distributed architecture called 2-LAMA (Two Level Assisted MAS Architecture) to endow an organisation with adaptation capabilities. In particular the architecture has two levels: a meta-level, where assistant agents (agents that determine how to adapt the regulations) are located, and a domain-level, the actual level where normal agents conduct their activities. They represent norms (if-then rules) via norm patterns, a specification that allows for dynamic run-time parametrization, and describe an adaptation mechanism based on case-based reasoning to learn from the behavior of agents. Adaptation is performed at run-time in two phases. In the first phase, each assistant agent individually performs a norm

adaptation. The decision on how to adapt norms is taken by each agent based on similar previously seen cases, using case-based reasoning (CBR), a machine learning approach based on the assumption that similar problems have similar solutions and thus similar states require similar regulations. Cases represent an abstraction of a concrete problem situation and can be seen as run-time data. In particular a state is composed by: the problem, a set of attributes describing the state of the system; the solution, a set of attributes describing the solution previously used in the problem, i.e., the parameters of the norms; and an evaluation representing how well the solution of the case was evaluated in the case. The norm revision corresponds to selecting for the current problem new norms. In the second phase, finally, the norms adapted by each assistant agent are collected and, via a voting mechanism, a final adaptation is approved. Once the new norms are enforced, they are monitored and the new cases are updated, by also taking into account the previous knowledge.

The approach presented by Miralles *et al.* [208] is conceptually close to some of the solutions proposed in this thesis. One of the major differences is that, instead of using CBR, we make use of a probabilistic (graphical) model (Bayesian Networks) to guide the revision. The graphical feature of a Bayesian Network allows us to use it as a run-time tool for maintaining alive requirements models, and therefore to *explicitly* validate their structural assumptions. Moreover, Bayesian Networks are considered transparent tools to inspection, falling under the umbrella of Explainable AI [58], and they facilitate requirements engineers in understanding the erroneous assumptions made at design-time. Another major difference is that, in particular in Chapter 5, we focus on conditional norms with deadlines, instead of if-then rules. While the latter allow to express either state-based or action-based norms, the former allow to express a broader set of behavioral rules.

Morales *et al.* [210, 211] present algorithms to synthesize more liberal and compact norms based on the behavior of the agents monitored at run-time. LION [210], for example, is an algorithm for the synthesis of *liberal* normative systems, i.e., that sets as few constraints as possible on the agents' actions. A graph structure called *normative network* is used to characterize the generalization relationship between different norms, in a similar fashion to a requirements model. The synthesis is guided by run-time data in terms of statistical evidence about the relationships between norms. For instance, data concerns whether two concurrently applicable norms are substitutable (if no conflict is monitored whenever one of them is fulfilled), or whether they are complementary (if some conflict is monitored whenever one of them is violated). The norm revision is then performed, based on such data and on the normative network, in order to obtain new norms that achieve properties of the normative systems such as its liberality or compactness. Similarly to Miralles *et al.* [208], the norms considered are state-based and action-based norms.

The concept of generalization of a norm described by Morales *et al.* relates with some of the norm revision operations that we propose in this dissertation (namely, weakening and strengthening). Weakening a norm generates more general norms, while strengthening generates more specialized ones. As we have seen earlier, similar operations have been described also in the software and requirements engineering literature, for example by Lamsweerde *et al.* [276] or by Kafali *et al.* [172], who define design patterns for the iterative revision and verification of a specification. In this

dissertation, revision is meant to align the enforced norms with the system's objectives, which are properties that are desired from the *behavior* of the whole system. Liberality and compactness aspects of the norms, which are not directly related to the behavior of the system, can be seen as a complementary extension of our proposals.

Corapi [94] and Athakravi [24] discuss the application of ILP to norm revision. In their work, the system designer describes desired properties of the the system through use cases (event traces associated to a desired outcome state), and they use ILP to revise the predefined norms so to satisfy the use-cases. In a similar way, Alrajeh *et al.* [18] propose to use ILP to infer requirements from a set of scenarios (the run-time data) that are obtained from the stakeholders of the system and describe desirable and undesirable behaviors of the system, and an initial (incomplete) requirements specification. To do so, they translate the available specification and the scenarios into an event-based logic programming formalism and use ILP to learn a set of missing event preconditions. Approaches using ILP to norm and requirements revision are related mainly to Chapter 5 of this dissertation, where we discuss how to revise conditional norms with respect to a given set of execution traces. ILP-based approaches and our proposal represent different trade-offs between the amount of background knowledge assumed about the possible causes of norm violations, and the guarantees that can be given regarding a particular (candidate) revision. In ILP-based approaches, the norms and the desired outcome of execution traces are expressed in the same language. This allows to directly modify the norms based on the desired outcomes (e.g., introducing a missing condition in the requirement). In our proposal, instead, we consider a type of desired objectives that cannot be directly enforced (e.g., it is not possible to directly enforce safety on vehicles: "no accidents should occur" is not directly enforceable on drivers). In this sense, we relax the assumption of being explicitly given with the causal relation between the norms and the system's objectives, which makes ILP-based approaches usable to generate provably correct norm revisions. Instead, we use statistical analysis to drive the revision of norms.

2.6 Discussion

In this chapter, we have discussed the key aspects concerning the run-time supervision of autonomous software systems executing in dynamic and weakly controllable settings with evolving objectives. We summarize here the major limitations that we identified in the state of the art, and that motivate the research questions outlined in Chapter 1 and the contributions of the following chapters.

- Several works have discussed the importance of explicitly considering assumptions made at design-time and of continuously monitoring them at run-time to assess their validity [15, 56, 191]. Only initial work, however, exists that characterizes the design-time assumptions reflected in the structure of a requirements model [16], a design artifact often used in self-adaptation contexts for keeping the requirements of the system alive at run-time [40, 271]. No run-time approach has been proposed to support the automatic probabilistic evaluation of assumptions in requirements models by means of execution data, and to use such evaluation for informing an automatic revision of requirements.

In Chapter 3, we address these limitations, and we propose the use of Bayesian Networks (BN) [240] as a run-time counterpart of a requirements model. While BNs have been widely employed in several fields for a variety of diagnostic and predictive tasks [115, 141, 147], no formal mapping with requirements models has been proposed before in the literature. We introduce an expressive type of BN and discuss its adoption to validate design-time assumptions, and to inform the automatic revision of requirements.

- The literature lacks of adaptation frameworks that are able to (provide suggestions to) revise the requirements. Most of the approaches that emerged in the context of self-adaptive systems and autonomic computing concern self-reconfiguration capabilities [142, 185, 289], including dynamic services composition [32]. A small part of the literature focused on self-adaptation at the requirements level by providing solutions for reassessing the priorities of requirements [40] and for selecting appropriate subsets of the requirements at run-time [44, 98]. The great majority of these solutions aims at guaranteeing an adequate compliance of the system with the given requirements. In these approaches, however, requirements are not questioned and the system's objectives are assumed to be achieved when the requirements are satisfied.

  Such a limitation is one of the major motivations of the work presented in this dissertation. For this reason, we present a run-time framework that uses data to continuously assess the effectiveness and usefulness of requirements in different operating contexts, and to revise the requirements when necessary. In Chapter 3, we focus on revising requirements models by selecting the most promising requirements variants. In Chapter 4, we extend the framework to support also the revision of sanctions associated to requirements' violation. In Chapter 5, we make a step further and we study how to automatically synthesize new requirements instead of relying on predefined ones.

- The great majority of the existing requirements and norms revision approaches relies on given domain knowledge, such as given utility functions [17, 41], or models of the system and of the agents that allow to evaluate possible alternative solutions (e.g., by means of model checking or via simulation) before putting them in place [59, 232]. When dealing, like in our case, with autonomous agents from the point of view of an exogenous institution, knowledge about the internals of the agents cannot be assumed. Also, estimating utilities for evolving objectives requires expensive, time-consuming human intervention. Explicit knowledge about the causal relation between requirements and system's objectives, like in the case of ILP-based approaches [94], is often not available.

  In the following chapters, we propose a number of domain-independent strategies for suggesting and performing automatic requirements revision. The proposed strategies are data-driven in the sense that they are based only on an explainable statistical analysis of data that is acquired during the execution of the system. By combining such strategies with an hill-climbing optimization approach, they can be directly applied at run-time and do not require additional domain knowledge, nor a model of the system or of the agents.

- Different approaches to the run-time revision of sanctions have been proposed over the years. The existing approaches, however, either rely on a model of the agents so that their behavior can be analyzed (e.g., via simulation) prior execution to determine optimal norms and sanctions [59], or they are based on the assumption of correct and effective norms, for example in cases where sanctions are determined proportionally to the number of run-time violations [72].

  In Chapter 4, we propose an alternative approach that does not rely on knowledge of the internals of the agents, opaque to our revision mechanism. Instead, we combine execution data with an estimation of the preferences of the agents, which can be obtained by observing the agents. Preferences in MASs have been used as a way for the agents to choose between different plans or actions to execute [103, 168, 223, 286], or within game theoretical frameworks, e.g., in the context of mechanism design [68]. We investigate the relationship between the estimation of the agents' preferences and their run-time behavior to effectively and quickly determine optimal sanctions in MASs. Furthermore, we relax the assumption that norm violation is detrimental, and we accommodate it by reducing the sanctions if data provides opposite evidence.

- The literature lacks approaches for the run-time synthesis of norms and requirements based on data. The few proposed approaches (i) are limited to state-based or action-based norms [207, 210]; (ii) they make strong assumptions if considered in the context of complex, weakly controllable, and evolving systems, e.g., ILP-based approaches assume that the desired outcomes of executions can be directly enforced via requirements and that the causal relation between the norms and the outcomes is given [24, 94]; (iii) they are concerned with goals which are not explicitly meant to guarantee system-level objectives, such as resolving conflicts between norms [213] or learning or identifying norms [95].

  In Chapter 5, we make a step to bridge this gap in the literature. We focus in particular on conditional norms (e.g., conditional obligations and prohibitions), a type of norms that aims at regulating patterns of behaviors instead of specific actions or states. Early work on approximation of conditional norms have been conducted in the context of monitor synthesis [10]. No approach, however, has been proposed to revise and synthesise conditional norms based on execution traces in order to align them with the system's objectives. In the chapter we contribute to the literature with theoretical results concerning the complexity of the problem, and with a novel approach for its approximate solution.

3 A Framework for the Supervision of Autonomous Systems

Complex modern systems such as Socio-Technical systems (STSs) are defined by the interaction between technical systems, like software and machines, and social entities, like humans and organizations. The entities within an STS are autonomous, thus weakly controllable, and the environment where the STS operates is highly dynamic. As a result, the designed requirements may end up being invalid and ineffective to guarantee the intended system's objectives when the system operates, for the autonomous entities do not comply with them, or the environment changes. In this chapter, we introduce a framework for the run-time validation and revision of the requirements of a software system. This chapter aims at bridging the gaps in the literature concerning (i) the lack of probabilistic mechanisms for the run-time validation of the assumptions expressed in requirements models, and (ii) the lack of a practical run-time supervision framework that continuously analyses and diagnoses the validity of requirements for STSs, and automatically suggests how to revise the requirements when there is evidence from data of their ineffectiveness. We use a Bayesian Network to collect run-time data and to learn the probability of achieving the system's objectives with different requirements in different operating contexts. We leverage the learned knowledge to determine which assumptions in a requirements model, a design artifact that represents the requirements of the system, are invalid, and to devise heuristic strategies for suggesting how to revise the requirements (i.e., whether to relax, strengthen or alter them). We evaluate the effectiveness of the supervision mechanism in selecting appropriate alternative configurations of requirements on a smart traffic scenario. The results show that our heuristics, informed by run-time execution data, outperform standard uninformed heuristics, in terms of convergence speed, solution quality, and stability. Moreover, the algorithms show good resilience to noise introduced into the execution data.

This chapter has been published in:

- Dell'Anna, Davide, Fabiano Dalpiaz, and Mehdi Dastani. "Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing". *Automated Software Engineering*, 26.3 (2019): 513-557.

We emphasise that in the context of this dissertation we interpret requirements as norms for a Normative Multi-Agent System. In this chapter, we refer to requirements as norms that are regimented, rather than enforced by means of sanctions. We leave considerations about sanction-based enforcement mechanisms for Chapter 4. Here, we study the effect of imposing different norms (requirements) on a Multi-Agent System (STS), their validity and relationship with the system's objectives in different contexts and the possible strategies to revise requirements when they are not effective. This chapter answers research question **RQ 1** from Chapter 1 and provides an answer to **RQ 2-4** in the context of revision of requirements models for STSs.

A preliminary version of this chapter has been published, in the context of Multi-Agent Systems, in:

- Dell'Anna, Davide, Mehdi Dastani, and Fabiano Dalpiaz. "Runtime norm revision using Bayesian networks". *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, Cham, 2018.

Algorithms 2 and 3 in Sec. 3.6.2 of this chapter, have been included, with their description, from the above work for completeness.

3.1 Introduction

For over forty years, researchers and practitioners in software and requirements engineering (RE) have proposed and experimented methods and tools to specify and evolve the requirements of *software systems* [191, 276]. However, the increasing embedding of cyber-physical and socio-technical systems (STSs) [81, 100, 258] in our lives poses new challenges for the RE discipline.

A smart city, for example, is an STS that includes heterogeneous entities such as pedestrians, drivers, vehicles, bicycles, traffic lights and signs, speed cameras, and road regulations. This STS is governed by the city council that can alter the road regulations and control artifacts such as traffic lights to best achieve the system's objectives (e.g., to reduce jams). However, many entities (humans and vehicles) are autonomous and therefore weakly controllable [81].

The autonomy of the participating entities and the dynamic, open nature of STSs [100, 258] entail that anticipating all the possible states of the system and transitions between them is not an option [190, 294] and the compliance of the system with its requirements cannot be guaranteed. Run-time requirements monitoring and diagnosis are therefore essential activities to determine system compliance with its requirements, which may eventually trigger evolution or adaptation mechanisms.

Several frameworks [140, 236, 290] have been proposed to support run-time requirement monitoring and diagnosis. Many of such approaches represent requirements via requirements models [40, 290], and analyze system execution data in terms of requirements satisfaction.

The self-adaptive systems literature goes beyond diagnosis, and proposes solutions to adapt a system when their requirements are threatened [41, 113, 185]. Self-adaptive systems search for a new system configuration that is expected to outperform the current one in achieving the system requirements.

Unfortunately, state-of-the-art approaches implicitly rely on the correctness of the requirements model. When designing a system, however, requirements engineers make *assumptions* about requirements, their satisfaction conditions, and the environment in which the requirements should be satisfied [15, 55, 56, 190, 191]. This is even more true for STSs, due to the autonomy of the participating entities and the volatility of the environment.

In this chapter, we propose a framework (see Fig. 3.1) for the adaptation and evolution of STSs that challenges the validity of the assumptions in a requirements model. We use Bayesian Networks to learn the relationship between the satisfaction of requirements and system's objectives. Based on such information, the framework can be used to (i) validate the assumptions in the model and let the analyst manually evolve the system or its model; and (ii) automatically revise the requirements model by determining the most appropriate requirements for the achievement of the system's objectives.

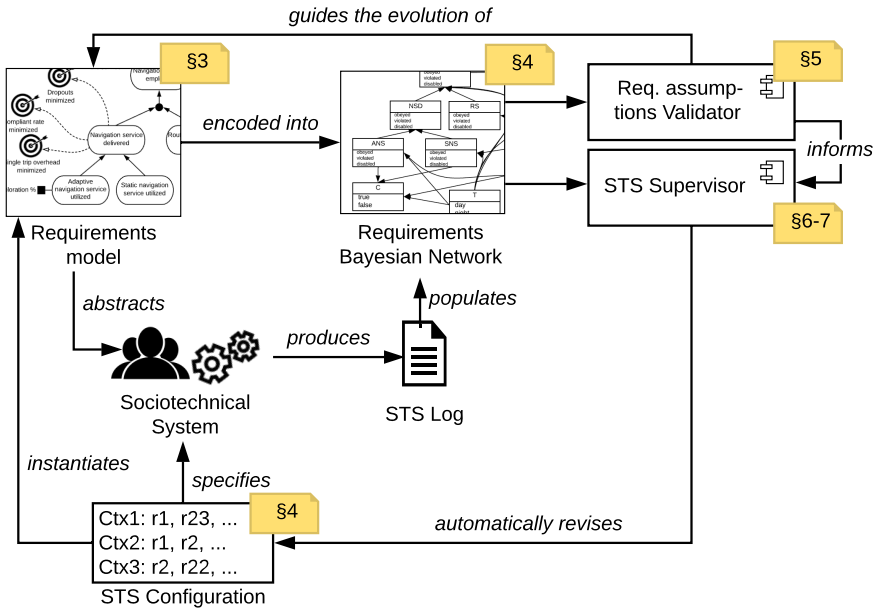


Figure 3.1: *Overview of the framework for STS evolution*

Specifically, we make the following contributions to the literature:

- We propose *Requirement Bayesian Networks* (*RBN*) as the run-time counterpart of the requirements models created at design-time; an *RBN* is populated with execution logs and apprehends the causal relationships between requirements and system's objectives in the different operating contexts;

- We explain how a human analyst can validate the design-time assumptions in a requirements model through the use of an *RBN*;

- We present an automated requirements revision mechanism that can be used for the system to identify sets of requirements that maximize the achievement of the system's objectives in each operating context. A first version of our heuristics was presented in [117], with a focus on identifying optimal norms to govern the behavior of a multi-agent system. The approach employs a variant of the hill climbing optimization technique to iteratively revise the norms based on their effectiveness in achieving the system's objectives. In this chapter, we extend this work along three dimensions: (i) we apply our heuristics to the case of hierarchical requirements models, as opposed to flat norm sets; (ii) we formally define the concepts of requirement variant, system configuration, and requirement revision; and (iii) we present a substantial evaluation of our algorithms;

- Via a smart traffic simulation applied to a mid-sized city, we evaluate how effective our revision mechanisms are at finding good-enough requirements.

Organization. Sec. 3.2 presents our research background. Sec. 3.3 introduces the smart traffic working example that we use throughout the chapter. Sec. 3.4 defines *RBNs* and shows how to map requirements models to them. Sec. 3.5 presents different types of design-time assumptions and describes how to validate them based on *RBN* information. Sec. 3.6 elaborates on our framework for automatic requirements revision. Sec. 3.7 reports on our evaluation using a smart traffic simulator. Sec. 3.8 reports on related work. Sec. 3.9 discusses our work and sketches future work.

3.2 Background

We present the key background for this chapter: (i) requirements models; and (ii) Bayesian Networks for representing and learning knowledge.

3.2.1 Requirements Models

Requirements models have been used and studied extensively in RE. As pointed out by the IREB handbook of requirements modeling [97], such models can be created with different purposes, including specifying a system, supporting testing, and increasing clarity. Depending on the purpose, the analysts may decide to represent the information structure, scenarios, goals and objectives, or other aspects of the system under development and its environment.

Here, we focus on hierarchical requirements models, that organize the requirements for a system as refinement trees, where high-level objectives—explaining the *raison d'être* for the requirements [303]—are specified in terms of more specific requirements and system functions. In particular, we take inspiration from the rich literature on goal models [101, 275, 303], but choose a general notation that does not commit to a specific modeling language.

A small requirements model for a car wash service is shown in Fig. 3.2. We distinguish between *requirements* and *objectives*. Requirements (rounded rectangles) define the behavior that the designer expects the entities within the STS to perform.

For example, having *cars cleaned*, or doing so via a *fully automated wash*. Objectives (targets with an arrow) express the conditions that denote stakeholder satisfaction with the system; for example, *Customer retention rate over 80% per year* indicates that the car wash owners do not simply want cars to be cleaned, but they aim at retaining most customers to sustain their business.

We organize requirements in hierarchies via AND- and XOR-*refinements*. For an AND-refined requirement to be satisfied, all of its sub-requirements need to be satisfied. For example, in order to have cars cleaned, both the interior and the exterior of cars should be cleaned, and positive opinions should be reported by the drivers. A XOR-refined requirement describes possible *mutually exclusive* ways for its achievement. For example, exterior cleaning can be done either via a fully automated wash or through a manual wash.

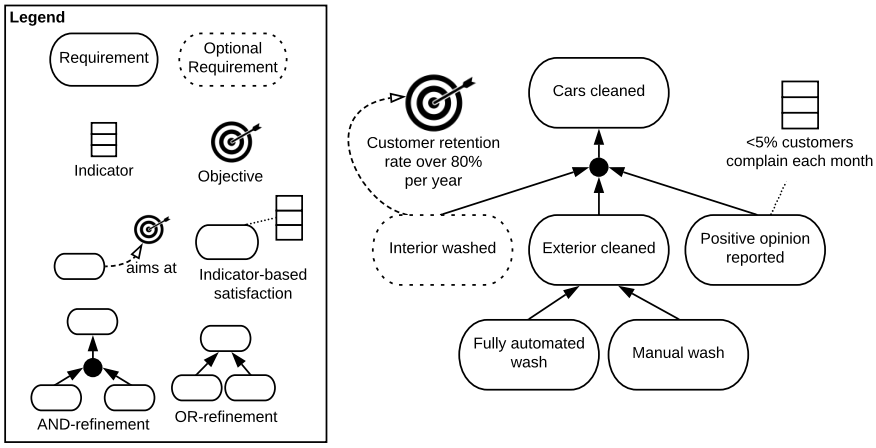


Figure 3.2: *A small requirements model for a car wash service.*

The expected impact of requirements on objectives is represented via *aims at* links, which denote positive contributions from requirements to objectives [154]. In Fig. 3.2, the designer expects that washing the car interior will support achieving an 80% customer retention rate.

We use *indicators* to qualify the satisfaction of requirements. Requirements that are not associated with an indicator (called regular requirements) are required to be satisfied by every instance of the said requirement. For example, *manual wash* is satisfied when all cars starting a manual wash are actually washed. Conversely, the satisfaction of requirements that are associated with an indicator (called aggregate requirements) is determined by aggregating a number of instances of that requirement. For example, *positive opinion reported* is satisfied when less than 5% of the customers complain. The analyst should specify the frequency for evaluating the indicator (e.g., monthly).

Finally, requirements can be optional (dotted border), indicating that they can either be selected or not selected. For example, *interior washed* is not necessary for having *cars cleaned*.

We formalize our requirements model in Def. 1, which is used in Sec. 3.4 to explain how requirements models are mapped to Bayesian Networks.

Definition 1 (requirements model)**.** *A requirements model is a tuple* $\mathcal{RM} = \langle(\mathcal{R}, ch, d), \mathcal{O}, \mathcal{SC}, cl, type, sc, opt\rangle$*, where*

- $(\mathcal{R}, ch, d)$ *is an AND-OR tree, where* $\mathcal{R} = \{R_1, ..., R_n\}$ *is a set of requirements,* $ch : \mathcal{R} \rightarrow 2^{\mathcal{R}}$ *is a function that returns the children of a requirement, and* $d : \mathcal{R} \rightarrow \{\texttt{AND}, \texttt{XOR}\}$ *is partial function that determines the type of refinement of a requirement with children;*
- $\mathcal{O} = \{O_1, ..., O_m\}$ *is a set of objectives;*
- $\mathcal{SC} = \{SC_1, ..., SC_{n+m}\}$ *is a set of satisfaction conditions for requirements, objectives, and indicators (see for instance Table 3.2);*
- $cl : \mathcal{R} \rightarrow 2^{\mathcal{O}}$ *is a function that maps requirements to the objectives that they aim at;*
- $type : \mathcal{R} \rightarrow \{\texttt{agg}, \texttt{reg}\}$ *is a function determining whether a requirement is aggregate or regular (all instances should be achieved);*
- $sc : \mathcal{R} \cup \mathcal{O} \rightarrow \mathcal{SC}$ *is a function that determines the satisfaction condition of requirements and objectives; and*
- $opt : \mathcal{R} \rightarrow \{\texttt{true}, \texttt{false}\}$ *is a function determining whether or not a requirement is optional.*

The requirements model in Fig. 3.2 can therefore be expressed according to Def. 1. A partial formalization is the following:

- $\mathcal{R} = \{$*cars cleaned*, ..., *positive opinion reported*$\}$
- $ch($*cars cleaned*$) = \{$*interior washed*, *exterior cleaned*, *positive opinion reported*$\}$, $ch($*exterior cleaned*$) = \{$*fully automated wash*, *manual wash*$\}$
- $d($*cars cleaned*$) =$ AND, $d($*exterior cleaned*$) =$ XOR
- $\mathcal{O} = \{$*customer retention rate over 80% per year*$\}$
- $cl($*interior washed*$) = \{$*customer retention rate over 80% per year*$\}$
- $opt($*interior washed*$) =$ true, ...

3.2.2 Bayesian Networks

Bayesian Networks have been widely used in many fields, ranging from medicine to forensics, as knowledge representation structures for learning and reasoning about the inter-dependencies between their nodes [240].

In software engineering, their applications include evaluating software reliability [127], estimating software effort [204], modeling software quality [209], and defect prediction [138]. In RE, Bayesian Networks have been employed both for the run-time verification of requirements [141] and for decision making [41].

Definition 2 (Bayesian Network)**.** *A Bayesian network [240]* $\mathcal{B} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$ *is a directed acyclic graph, where:*

- $\mathcal{X}$ *is the set of all nodes, each corresponding to a random variable in probability theory with a discrete or continuous domain (i.e., the set of possible values the node can take).*
- $\mathcal{A}$ *is the set of directed links (arrows) connecting pairs of nodes. If there is an arrow from node* X *to node* Y*,* X *is said to be a parent of* Y*. The set of parents of a node* Y *is denoted as* $\boldsymbol{Parents}(Y)$*.*
- $\mathcal{P}$ *is a set of* $|\mathcal{X}|$ *conditional probability distributions. Each node* $X \in \mathcal{X}$ *is associated with a conditional probability distribution* $\boldsymbol{P}(X|\boldsymbol{Parents}(X))$ *that quantifies the effect of the parents on the node.*

Note that in the context of Bayesian Networks we use the notation shown in Table 3.1. The pair $(\mathcal{X}, \mathcal{A})$ is called the *structure* of the Bayesian Network $(\mathcal{X}, \mathcal{A}, \mathcal{P})$. An evidence $\mathbf{e}$ is a revealed (observed) assignment of values for some or all of the random variables in the Bayesian Network, i.e., $\mathbf{e} = \{X_v | X \in \mathbf{X}\}$ with $\mathbf{X} \subseteq \mathcal{X}$ and v a possible value in the domain of the variables.

Table 3.1: *A summary of the notation used for Bayesian Networks.*

Notation	Description
X, Y, ...	Random variables (italic uppercase)
$\mathbf{X}$, $\mathbf{Y}$, ...	Set of random variables (bold uppercase)
v_1, v_2, ...	Value in the domain of a random variable (italic lowercase)
$\mathbf{x}$, $\mathbf{y}$, ...	Assignment of values to a set of nodes (bold lowercase)
X_v	$(X = v)$, assignment of value v to a random variable X
$\mathbf{X}_v$	Assignment of value v to all nodes in $\mathbf{X} \subseteq \mathcal{X}$
X_{act}	$\neg X_{dis} = \neg(X = disabled)$, the fact: X is not *disabled*
$\mathbf{P}$	Probability distribution
P	Single probability

Given the set $\mathcal{X}$ of all the nodes in a Bayesian Network $\mathcal{B}$ and a (possibly empty) evidence $\mathbf{e}$, reasoning with $\mathcal{B}$ generally means to determine the distribution $\mathbf{P}(\mathbf{X}|\mathbf{e})$, with $\mathbf{X} \subseteq \mathcal{X}$ a set of nodes of which we want to discover the probability distribution (e.g., $\mathbf{P}(X|Y_v)$ is the probability distribution of the values of the random variable X, given that value v is observed for variable Y).

3.3 The CrowdNavExt Smart Traffic Simulator

In this chapter, we study the evolution of an STS through computer simulation, a powerful tool for testing alternative configurations prior to changing the real environ-

ment, which is particularly adequate to analyze the behavior of autonomous agents in a large-scale real setting [198, 268, 300].

We start from the CrowdNav *smart traffic simulator*, an exemplar from the self-adaptive systems literature [245] that simulates traffic scenarios in the middle-sized city of Eichstädt, in Germany, with 450 streets and 1,200 intersections. We propose CrowdNavExt[1], which introduces multiple types of navigation services as well as different ways of managing junctions, in line with the requirements model of Fig. 3.3, described in the following.

The city council of Eichstädt aims at improving the traffic by achieving two system's objectives: ensuring an *average trip overhead below 250%* compared to the theoretical traveling time without traffic, and guaranteeing *less than 4 accidents per day*. To achieve such objectives, the city council plans to opportunely manage junctions in the city and to offer to the cars a Centralized Navigation Service (CNS) in addition to the cars' personal navigation system. Due to the highly dynamic nature

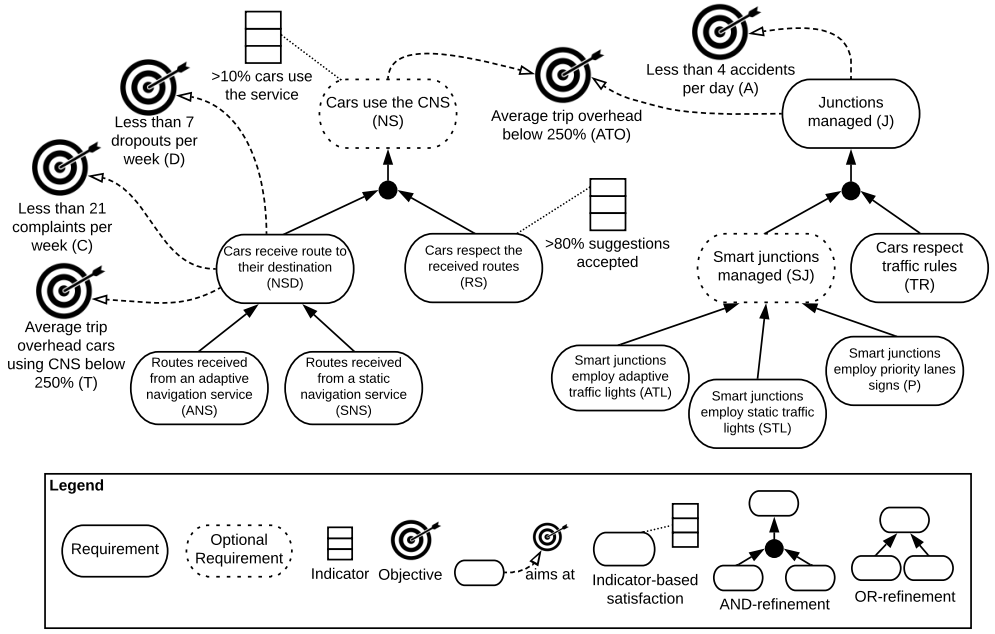


Figure 3.3: *A requirements model for the smart traffic simulation.*

of the city, drivers and vehicles can behave differenlty in different contexts. In this chapter, we consider two *contextual properties* (*Time* and *Weather*), that can assume two values each: *Time* can either take *day* or *night*, while *Weather* can either be *normal* or *extreme*.

Two top-level requirements, set by the city council to achieve the objectives, are the following: *at least 10% of the cars in the city shall always use the offered CNS*

[1]CrowdNavExt's code repository: https://bitbucket.org/dellannadavide/crowdnavext.

(the requirement *NS* in Fig. 3.3 and the associated indicator), and *every junction in the city is opportunely managed* (requirement J).

To satisfy the requirement *NS*, two sub-requirements are *assumed* to be necessary: *whenever a car starts a trip toward a destination, the car shall receive a route from the Central Navigation Service* (*NSD*) and *at least 80% of all the route suggestions given by the CNS are respected by the cars equipped with the CNS* (*RS*). *NSD* can be met by either employing a self-adaptive navigation service (*ANS*) [245] or a static navigation service (*SNS*). In our simulator, each car relies on a navigation service to determine its route from origin to destination: 90% of the vehicles use their personal navigation service (the default routing algorithm of the simulator), while the remaining 10% are smart cars that can use a centralized navigation service. When smart cars do not use the centralized navigation service, they use their own navigator as normal cars.

The *NSD* requirement is assumed to help achieve three additional objectives concerning the satisfaction of the users of the navigation service: *less than 21 complaints per week* (C); *less than 7 dropouts per week* (D), i.e., cars that decide to stop using the CNS; and *average trip overhead cars using CNS below 250%* (T), for some cars using the CNS will be suggested paths to explore in order for the CNS to identify optimal paths.

To satisfy the requirement J, two sub-requirements are assumed to be necessary: *every junction that is equipped with smart panels (called smart junctions) shall display on the panel the prescribed traffic rule* (*SJ*), and *every car shall respect the traffic rules prescribed by the junctions in the city* (*TR*). *SJ* can be met by either displaying on the panels traffic lights that adapt their timing according to the traffic (*ATL*), or by displaying regular traffic lights (*STL*), or by displaying which of the lanes in the junctions has priority (P). When no management is prescribed for smart junctions, the vehicles approaching the junctions follow the default priority-to-the-right rule.

Table 3.2 describes precisely the conditions for requirement monitors to determine requirements and objectives satisfaction.

3.4 From Requirements Models to Bayesian Networks

In this section, we define the type of Bayesian Network (called *Requirement Bayesian Network*, or *RBN*) that we use for supporting requirements evolution, and we explain how to automatically generate the structure of an *RBN* from requirements models as presented in Def. 1.

3.4.1 Requirement Bayesian Network

Let $\mathcal{CP} = \{\mathcal{CP}_i, ..., \mathcal{CP}_k\}$ be a set of monitorable contextual properties of the STS system (i.e., monitorable environmental variables that determine the operating context of the system, e.g., *Time*, *Weather*), each associated with a domain of values (e.g., *Weather* can be either *normal* or *extreme*).

Definition 3 (Requirement Bayesian Network)**.** *A Requirement Bayesian Network* $\mathcal{RBN} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$ *is a Bayesian Network where:*

Table 3.2: *Satisfaction conditions of the requirements and objectives in our scenario.*

Var	Satisfied	Eval. every
NS	$> 10\%$ vehicles in the city is using the CNS	time instant
NSD	every time a CNS-equipped car starts a trip, it receives a route from the CNS	trip
ANS	every time a CNS-equipped car starts a trip, it receives a route from an ANS	trip
SNS	every time a CNS-equipped car starts a trip, it receives a route from a SNS	trip
RS	$> 80\%$ of all CNS suggestions has been accepted	week
J	all sub-requirements are satisfied	time instant
SJ	every smart junction displays the traffic rules on its panel	time instant
ATL	every smart junction displays adaptive traffic lights	time instant
STL	every smart junction displays regular traffic lights	time instant
P	every smart junction displays priority lanes signs	time instant
TR	every time a car crosses a junction, it satisfies the displayed traffic rule	car at junction
ATO	the average trip overhead of all the vehicles has been below 250%	week
A	the number of accidents is below 28	week
C	the number of complaints received is below 30	week
D	the number of dropouts is below 7	week
T	the average trip overhead of vehicles using the CNS has been below 250%	week

- $\mathcal{X} = \boldsymbol{R} \cup \boldsymbol{O} \cup \boldsymbol{C}$ *is a set of nodes, representing random variables in probability theory. The sets* $\boldsymbol{R}$, $\boldsymbol{O}$ *and* $\boldsymbol{C}$ *are disjoint.*
  - $\boldsymbol{R}$ *consists of requirement nodes. Each node* $R \in \boldsymbol{R}$ *corresponds to a requirement and has a discrete domain of 3 possible values: obeyed, violated and disabled.*
  - $\boldsymbol{O}$ *consists of objective nodes. Each node* $O \in \boldsymbol{O}$ *corresponds to a boolean objective and has a discrete domain of 2 values: true and false.*
  - $\boldsymbol{C}$ *consists of context nodes. Each node* $C \in \boldsymbol{C}$ *corresponds to a contextual property* $\mathcal{CP}_i \in \mathcal{CP}$ *and can have discrete or continuous domain.*
- $\mathcal{A} \subseteq (\boldsymbol{R} \times \boldsymbol{R}) \cup (\boldsymbol{C} \times \boldsymbol{R}) \cup (\boldsymbol{C} \times \boldsymbol{O}) \cup (\boldsymbol{R} \times \boldsymbol{O})$ *is the set of arrows connecting pairs of nodes. If there is an arrow from node* X *to node* Y*,* X *is said to be a parent of* Y*.*
- $\mathcal{P}$ *is a set of conditional probability distributions, each one associated with a node in* $\mathcal{X}$ *and quantifying the effect of the parents on the node.*

An evidence $\mathbf{c}$ for all the context nodes $\mathbf{C}$ is an observation for a certain context

(e.g., *Time* has value *day* and *Weather* has value *normal*). For simplicity, we call *context* also the associated evidence in the RBN.

Note that when we refer to nodes of a specific type, unless otherwise specified, we use the corresponding notation convention, e.g., R refers to a node in $\mathbf{R}$, $\mathbf{c}$ refers to an assignment of values of nodes in $\mathbf{C}$, $\mathbf{R}_{viol}$ refers to an assignment of value *violated* to a set of requirement nodes $\mathbf{R}$, etc..

3.4.2 From Requirements Models to Requirement Bayesian Networks

We introduce the function *RM2BNS* that generates the *structure* of an $\mathcal{RBN}$ (Def. 3) from a requirements model $\mathcal{RM}$ (Def. 1). *RM2BNS* maps a requirements model $\mathcal{RM}$ and a set of contextual properties $\mathcal{CP}$ to an $\mathcal{RBN}$ structure $(\mathcal{X}, \mathcal{A})$. Note that the probability distributions $\mathcal{P}$ of an $\mathcal{RBN}$ do not depend on the source requirements model, but are populated from the system's execution log at run-time. Therefore, $\mathcal{P}$ will not be considered in this section.

As a preliminary notion, we denote the set of requirements contributing to (aiming at) an objective O in $\mathcal{RM}$ as $cont\_desc(O)$. A requirement R contributes to an objective O if R is a descendant of O and it is either a leaf requirement, or it is of type *aggregate* but has no ancestor of type *aggregate*:

$$\begin{aligned} &cont\_desc(O) = \\ &(desc(O) \setminus \{R' | R' \in desc(R), R \in desc(O), type(R) = \texttt{agg}\}) \setminus \\ &\{R | ch(R) \neq \varnothing, type(R) \neq \texttt{agg}\} \end{aligned} \tag{3.1}$$

where $desc(R)$ is the set the descendant of R (similar for O). For instance, in the requirements model of Fig. 3.3, $cont\_desc(D) = \{ANS, SNS\}$.

Definition 4 (RM2BNS). *Given a requirements model $\mathcal{RM} = \langle(\mathcal{R}, ch, d), \mathcal{O}, \mathcal{SC}, cl, type, sc, opt\rangle$ and a set of contextual properties $\mathcal{CP}$, the function RM2BNS returns the structure of a Requirement Bayesian Network; formally, $RM2BNS(\mathcal{RM}, \mathcal{CP}) = (\mathcal{X}, \mathcal{A})$, where*

- $\mathcal{X} = \mathcal{R} \cup \mathcal{O} \cup \mathcal{CP}$
- $\mathcal{A} = \{(R, O) \mid R \in cont\_desc(O)\} \cup$
  $\{(R_1, R_2) \mid R_1 \in ch(R_2)\} \cup$
  $\{(C, R) \mid C \in \mathcal{CP}, R \in \mathcal{R}, (type(R) = agg \vee \ ch(R) = \varnothing\ )\} \cup$
  $\{(C, O) \mid C \in \mathcal{CP}, O \in \mathcal{O}\}$

Intuitively, $\mathcal{A}$ contains (i) an arrow from a requirement node R to an objective O if R is a contributing descendant of O (see function *cont_desc* above), (ii) an arrow from a sub-requirement R_1 to its parent requirement R_2, (iii) an arrow from a context node C to each requirement node R that represents either a leaf requirement ($ch(R) = \varnothing$) or a requirement with an indicator, and (iv) an arrow from each context node C to each objective O.

Fig. 3.4 reports the structure of the $\mathcal{RBN}$ that is generated by applying Def. 4 to the requirements model of Fig. 3.3.

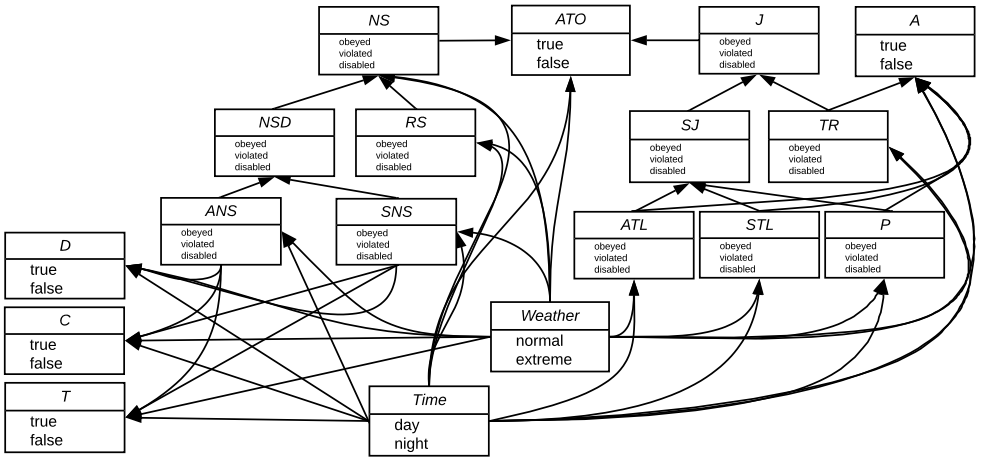


Figure 3.4: *The $\mathcal{RBN}$ structure defined by RM2BNS applied to the requirements model of Fig. 3.3.*

Besides reflecting the requirements model's topology, the network also introduces the context variables. The resulting structure of the *Requirement Bayesian Network*, which consists of requirement, objective and context nodes, allows to analyze the assumptions in different operating contexts (see Sec. 3.5). In an $\mathcal{RBN}$, every context node is parent of all the objective nodes. This indicates that the achievement of objectives is not only due to the satisfaction (or presence) of requirements, but also to events that occur in the environment. Context nodes are also parents of all the requirement nodes whose satisfaction is not exclusively determined by their hierarchical structure in the AND/OR tree, but can also be affected by the context in which they are applied.

We choose a three-values discrete domain for the requirement nodes to make the network more versatile: while the *obeyed* and *violated* values allow to evaluate assumptions about the satisfaction or violation of requirements (e.g., the *requirement satisfiability assumption* in Sec. 3.5), the *disabled* value supports XOR-refined requirements. To update the conditional probability distribution of a node, it is necessary to provide evidence for both the node and all of its parents. In case of a XOR-refined requirement, we obtain evidence only for one of the parents (sub-requirements) at a time. The *disabled* value allows therefore to perform the update also in such case.

3.4.3 Populating the *RBN*: Data collection

Table 3.3 reports a sample dataset that can be obtained from monitoring the requirement and objective satisfaction from the log for the working example of Sec. 3.3. The values that each of the variables assumes belongs to its domain as specified in Sec. 3.4.1 (e.g., *obeyed*, *violated*, *disabled* for requirement nodes, *true* or *false* for objective nodes). Such dataset can be used to train the *RBN* of Fig. 3.4 and learn the set of conditional probability distributions $\mathcal{P}$.

Table 3.3: *Part of the dataset used to train the BN of Fig. 3.4 and obtained from monitoring the execution of the system in Sec. 3.3.*

Weather	*Time*	*NS*	*NSD*	*ANS*	*SNS*	*RS*	*J*	*SJ*	*ATL*	*STL*	*P*	*TR*	*ATO*	*A*	*C*	*D*	*T*
norm	night	viol	ob	dis	ob	viol	viol	viol	dis	dis	viol	ob	T	T	T	T	F
norm	day	ob	ob	ob	dis	ob	ob	ob	ob	dis	dis	ob	F	F	T	F	T
norm	day	ob	ob	viol	dis	viol	viol	dis	viol	dis	viol	ob	F	F	T	F	F
extr	night	viol	ob	dis	ob	ob	viol	viol	dis	viol	dis	ob	T	T	T	T	F
extr	day	ob	dis	dis	dis	dis	ob	ob	ob	dis	dis	ob	T	F	T	T	F
extr	day	ob	ob	dis	ob	viol	viol	viol	dis	dis	viol	ob	T	F	T	T	T
...																	

A discussion of learning techniques (e.g., classical Bayesian learning) is out of the scope of this chapter; we refer the interested reader to the existing literature [240, 261]. Also, we do not analyze requirements monitoring mechanisms (e.g., EEAT [236]). In the following, we assume to have a trained *RBN*.

3.5 Design-Time Assumptions and Their Validation

Requirements models contain *assumptions* that might be, or become, invalid in practice [15, 190, 191]. In this section, we describe six types of assumptions made by the designer of a system during the definition of a requirements model (as per Def. 1), and we propose a mechanism to determine the validity of such assumptions by using an *RBN* trained with system execution data. As shown in Fig. 3.1, this information can be used by the analysts to guide the evolution of an STS.

We introduce the notion of *degree of validity* (δ in the following) for an assumption as a real number in the range $[-1, +1]$. $\delta = +1$ denotes a fully valid assumption, $\delta = -1$ indicates a fully incorrect assumption, and the intermediate values describe an assumption with partial validity.

δ is computed as a difference between two probabilities, representing the collected positive and negative evidence for the validity of that assumption, respectively. Thus, if the collected positive evidence is close to 1 and the negative evidence is close to 0, δ will be close to $+1$. Values around 0 show that the assumption is only partly valid since the positive and negative evidences for the validity have similar strength.

3.5.1 Types of design-time assumptions

We take as a baseline the types of assumptions by Ali *et al.* [15] and extend the list to support the structure of our requirements models. Note that the assumptions defined below are made *implicitly* by defining the structure of a requirements model. Therefore, even though they can be associated with a certain element of the requirements model (e.g., with an arrow or a node of the model), they are not explicitly represented in the model (unlike, e.g., the work by Boness and colleagues [56]).

Requirement satisfiability assumption. The hypothesis that in a specific operating context, a requirement is satisfied (e.g., *in context day-extreme, the requirement RS is satisfied*). Fig. 3.3 contains 11 requirement satisfiability assumptions (each one associated with a requirement) for each of the four possible operating contexts.

Given a context **c** and a requirement node R, the degree of validity of the associated requirement satisfiability assumption in context **c** is

$$\delta_S(R, \mathbf{c}) = P(R_{ob} \mid \mathbf{c}) - P(R_{viol} \mid \mathbf{c}) \tag{3.2}$$

Objective achievement assumption. The hypothesis that in a specific operating context, an objective is achieved (e.g., *in context day-extreme, the objective ATO is achieved*). Fig. 3.3 contains 5 objective achievement assumptions (each one associated with an objective) for each of the four possible operating contexts.

Given a context **c** and an objective node O, the degree of validity of the associated objective achievement assumption in context **c** is

$$\delta_O(O, \mathbf{c}) = P(O_{true} \mid \mathbf{c}) - P(O_{false} \mid \mathbf{c}) \tag{3.3}$$

Contribution assumption. The hypothesis that in a specific operating context, there is a positive synergy between the satisfaction of a requirement and the achievement of an objective connected via an *aims at* link (e.g., *in context day-extreme, there is a positive synergy between the satisfaction of requirement NS and the achievement of the objective ATO*). Fig. 3.3 contains 6 contribution assumptions (each one associated with an *aims at* link) for each operating context.

Given a context **c**, a requirement node R and an objective node O, the degree of validity of a contribution assumption is:

$$\delta_C(O, R, \mathbf{c}) = P(O_{true} \mid R_{ob} \wedge \mathbf{c}) - P(O_{true} \mid R_{viol} \wedge \mathbf{c}) \tag{3.4}$$

Notice that the degree of validity of *negative* contribution assumptions, if considered in the requirements model (omitted in this chapter), due to the boolean nature of the objective nodes, can be calculated as $-\delta_C$.

Refinement assumption. The hypothesis that in a specific operating context, the satisfaction of an AND-refined requirement depends on the satisfaction of all its sub-requirements (e.g.,*in context day-extreme, to satisfy the requirement NS both the requirements NSD and RS shall be satisfied*), and a XOR-refined requirement is satisfied only when one and only one of its sub-requirements is satisfied (e.g.,*in context day-extreme, the requirement NSD is satisfied when either ANS or SNS are satisfied*). Fig. 3.3 contains 2 AND-refinement assumptions and 2 XOR-refinement assumptions (each one associated with a refinement) for each of the 4 operating contexts.

Given a context **c**, a requirement node R and the set $\mathbf{R'} \in \mathbf{R}$ of its requirement nodes parents, let **r** be the disjunction of all possible assignments of values to variables in **R'** excluding the assignment $\mathbf{R'}_{ob}$, let **r1ob** be the disjunction of all possible assignments of values to variables in **R'** such that only one variable takes value *obeyed*, and let **ro** be the disjunction of all possible assignments of values to variables in **R'** excluding the assignments in **r1ob**.

$$\delta_{AND}(R, \mathbf{c}) = P(R_{ob} \mid \mathbf{R'}_{ob} \wedge \mathbf{c}) - P(R_{ob} \mid \mathbf{r} \wedge \mathbf{c}) \tag{3.5}$$

$$\delta_{XOR}(R, \mathbf{c}) = P(R_{ob} \mid \mathbf{r1ob} \wedge \mathbf{c}) - P(R_{ob} \mid \mathbf{ro} \wedge \mathbf{c}) \tag{3.6}$$

For example, the degree of validity of the AND-refinement assumption of the requirement *NS* in Fig. 3.3 unfolds as follows:

$$\delta_{AND}(NS, \mathbf{c}) = P(NS_{ob} \mid NSD_{ob} \wedge RS_{ob} \wedge \mathbf{c}) - P(NS_{ob} \mid \neg(NSD_{ob} \wedge RS_{ob}) \wedge \mathbf{c}) \tag{3.7}$$

Adoptability assumption. The hypothesis that in a specific operating context, there is a positive synergy between the satisfaction of a requirement and the satisfaction of each one of its sub-requirements separately (e.g., *in context day-extreme, there is a positive synergy between the satisfaction of the requirement SNS and the satisfaction of the requirement NSD*). Fig. 3.3 contains 9 adoptability assumptions (each one associated with a link between a sub-requirement and a requirement) for each operating context.

Notice that, while refinement assumptions concern *one-to-many* relationships (i.e., between one requirement and all of its children), adoptability assumptions concern *one-to-one* relationships (i.e., between a requirement and each of its sub-requirements separately).

Given a context $\mathbf{c}$ and two requirement nodes R and R' such that R' is parent of R, the degree of validity of the associated adoptability assumption in context $\mathbf{c}$ is

$$\delta_{AD}(R, R', \mathbf{c}) = P(R_{ob} \mid R'_{ob} \wedge \mathbf{c}) - P(R_{ob} \mid R'_{viol} \wedge \mathbf{c}) \tag{3.8}$$

Requirement necessity assumption. The hypothesis that in a specific operating context, the activation of a specific requirement is necessary condition for achieving all the objectives (e.g., *in context day-extreme, to achieve the five objectives ATO, A, C, D, T together, the requirement ANS must be activated*). Fig. 3.3 contains 11 requirement necessity assumptions (each one associated with a requirement) for each operating context.

This assumption concerns the *activation* of a requirement, regardless of its satisfaction; i.e., it is the hypothesis that, in order to achieve the objectives, it is better to keep active a requirement rather than disabling it.

Given a context $\mathbf{c}$, a requirement node R and a set of objective nodes $\mathbf{O}$, the degree of validity of the associated requirement necessity assumption in context $\mathbf{c}$ is

$$\delta_{N}(R, \mathbf{O}, \mathbf{c}) = P(\mathbf{O}_{true} \mid R_{act} \wedge \mathbf{c}) - P(\mathbf{O}_{true} \mid R_{dis} \wedge \mathbf{c}) \tag{3.9}$$

3.5.2 Validating assumptions

The requirements model of Fig. 3.3, despite its simplicity, contains 184 assumptions (46 for each operating context, as described above) that the requirements engineer who constructed it has implicitly made. This calls for automated mechanisms that assist requirements engineers in validating such many assumptions.

Table 3.4 reports an evaluation of the validity of the assumptions for the requirements model of Fig. 3.3 when executing the simulator of Sec. 3.3. Specifically, we ran the simulator in all the operating contexts and we collected from the simulation logs a dataset of about 4.6 millions rows, part of which is reported in Table 3.3. We

created an *RBN* for our scenario using the mapping function *RM2BNS*; this led to the network shown in Fig. 3.4. Then, we trained such network using the dataset obtained from the smart traffic simulation. For the learning, we relied on the functionality offered by the bnlearn R package [246]. At this stage, we could evaluate the assumptions.

The calculated degree of validity of the assumptions underlying the requirements model can be used by the designer of the system as a support to determine how to evolve the STS. We provide some examples from Table 3.4.

The objective *ATO* is hardly achieved in context *day-normal*, i.e., the degree of validity of the objective achievement assumption $\delta_O(ATO, \mathbf{dn})$ is below 0 (-0.273). This happens because of the higher number of vehicles driving during the day. The designer may therefore introduce different requirements to help achieve the objective, for instance, replacing the current centralized navigation service with a more intelligent one, or by changing the environment, e.g., by closing some roads to traffic.

Also, requirement *ANS* is harmful in context *night-normal*: the degree of validity of the requirement necessity assumption $\delta_N(ANS, \mathbf{O}, \mathbf{nn})$ is -0.7867, indicating that using *ANS* is detrimental to satisfying the objectives, which are quite positively satisfied when *ANS* is not employed. In the simulator, this happens for the adaptive navigation service uses some vehicles as "explorers" to find less congested roads [245]. This strategy appears to be harmful during the night since less vehicles drive in the city and roads are not congested. The designer may therefore disable the navigation service in such context.

The degrees of validity listed in the table can be also visualized directly on the original requirements model using a color overlay (see [118] for an example of such visualization). This may help the designer to quickly analyze the behavior of the system and to determine whether an intervention is required.

3.6 Automated Requirements Revision

In Sec. 3.5, we have described mechanisms for analysts to determine—assisted by an *RBN* that is populated with system execution logs— the validity of the assumptions that a requirements model implicitly contains. Such techniques help the analysts identify systems' behaviors that are not aligned with expectations, so that human evolution of the system requirements can be made.

Here, we present a control loop for the automated adaptation of an STS (Sec. 3.6.2), which leverages the information concerning assumptions validity learned at run-time, in order to revise the STS requirements aiming to maximize the system's objectives achievement. Prior to explaining the control loop, we define in Sec. 3.6.1 some key terms that concern our conceptual framework.

3.6.1 Requirement Variant, System Configuration, and Requirement Revision

The adaptation mechanisms presented in this section require the introduction of three basic notions: those of a *requirement variant* (Def. 5), *system configuration* (Def. 6)

Table 3.4: *The degree of validity of the assumptions made in Fig. 3.3 in the four different operating contexts day-normal weather (**dn**), day-extreme weather (**de**), night-normal weather (**nn**), night-extreme weather (**ne**).*

Assumption	**c = dn**	**c = de**	**c = nn**	**c = ne**
$\delta_S(NS, \mathbf{c})$	0.0580	0.1073	0.0473	0.0676
$\delta_S(NSD, \mathbf{c})$	0.1007	0.0983	0.0988	0.0946
$\delta_S(ANS, \mathbf{c})$	0	0	0	-0.0001
$\delta_S(SNS, \mathbf{c})$	0.1003	0.0980	0.0989	0.0940
$\delta_S(RS, \mathbf{c})$	0.0596	0.0598	0.0582	0.0581
$\delta_S(J, \mathbf{c})$	0	0	0	0
$\delta_S(SJ, \mathbf{c})$	0	0	0	0
$\delta_S(ATL, \mathbf{c})$	0	0	0	0
$\delta_S(STL, \mathbf{c})$	0	0	0	0
$\delta_S(P, \mathbf{c})$	0	0	0	0
$\delta_S(TR, \mathbf{c})$	0	0	0	0
$\delta_O(ATO, \mathbf{c})$	-0.2730	0.6591	0.9998	0.9999
$\delta_O(A, \mathbf{c})$	0.1119	-0.0379	0.8374	0.8181
$\delta_O(C, \mathbf{c})$	0.5079	0.5954	0.9605	0.9998
$\delta_O(D, \mathbf{c})$	0.9933	1	0.9998	0.9998
$\delta_O(T, \mathbf{c})$	0.3155	0.5604	0.7472	0.9737
$\delta_C(ATO, NS, \mathbf{c})$	-0.0744	-0.2470	0.0002	0
$\delta_C(ATO, J, \mathbf{c})$	0	0	0	0
$\delta_C(A, J, \mathbf{c})$	0	0	0	0
$\delta_C(C, NSD, \mathbf{c})$	-0.0137	0.3960	0.5105	0.4777
$\delta_C(D, NSD, \mathbf{c})$	0.4337	0.5454	0.5383	0.4998
$\delta_C(T, NSD, \mathbf{c})$	0.0704	0.3773	0.2412	0.5180
$\delta_{AND}(NS, \mathbf{c})$	0.0187	0.0226	0.0362	-0.0376
$\delta_{AND}(J, \mathbf{c})$	0	0	0	0
$\delta_{XOR}(NSD, \mathbf{c})$	0.1003	0.0980	0.0988	0.0940
$\delta_{XOR}(SJ, \mathbf{c})$	0	0	0	0
$\delta_{AD}(NS, NSD, \mathbf{c})$	-0.1008	-0.1033	-0.1008	-0.0019
$\delta_{AD}(NS, RS, \mathbf{c})$	0.0025	0.0043	0.0025	0.0025
$\delta_{AD}(NSD, ANS, \mathbf{c})$	0.3542	-0.1548	0.3542	-0.0253
$\delta_{AD}(NSD, SNS, \mathbf{c})$	1	1	1	0.6684
$\delta_{AD}(J, SJ, \mathbf{c})$	0	0	0	0
$\delta_{AD}(J, TR, \mathbf{c})$	0	0	0	0
$\delta_{AD}(SJ, ATL, \mathbf{c})$	0	0	0	0
$\delta_{AD}(SJ, STL, \mathbf{c})$	0	0	0	0
$\delta_{AD}(SJ, P, \mathbf{c})$	0	0	0	0
$\delta_N(NS, \mathbf{O}, \mathbf{c})$	-0.0068	-0.0029	-0.0022	0
$\delta_N(NSD, \mathbf{O}, \mathbf{c})$	-0.0040	-0.0056	0.0017	0.0029
$\delta_N(ANS, \mathbf{O}, \mathbf{c})$	-0.1003	-0.2498	-0.7867	-0.7336
$\delta_N(SNS, \mathbf{O}, \mathbf{c})$	-0.0033	-0.0036	-0.0005	0.0024
$\delta_N(RS, \mathbf{O}, \mathbf{c})$	-0.0018	0.0075	0.0018	-0.0015
$\delta_N(J, \mathbf{O}, \mathbf{c})$	0.1006	0.2502	0.7870	0.8967
$\delta_N(SJ, \mathbf{O}, \mathbf{c})$	-0.0005	-0.0008	0	0.0005
$\delta_N(ATL, \mathbf{O}, \mathbf{c})$	0.0093	-0.2501	-0.7865	-0.8968
$\delta_N(STL, \mathbf{O}, \mathbf{c})$	0.0004	0.0003	0	0.0003
$\delta_N(P, \mathbf{O}, \mathbf{c})$	0.0005	0.0013	-0.7864	0.0001
$\delta_N(TR, \mathbf{O}, \mathbf{c})$	0.1003	0.2501	0.7867	0.6191

and *requirement revision* (Def. 7).

Definition 5 (requirement variant)**.** *Consider a set $\mathcal{O}$ of stakeholders objectives in a requirements model $\mathcal{RM}$, and a set $\mathcal{C}$ of all possible contexts in which the system operates. We call requirement variant V a sub-graph of $\mathcal{RM}$ that is defined by pruning $\mathcal{RM}$ as follows:*

1. *for every XOR-refined requirement in $\mathcal{RM}$, V contains exactly one sub-requirement;*
2. *for every optional requirement in $\mathcal{RM}$, that requirement can either be included in or excluded from V.*
3. *if a requirement in $\mathcal{RM}$ is excluded from V through clauses 1. or 2., then all the descendants of that requirement are also pruned.*

The requirements model of Fig. 3.3 results in a set $\mathcal{V}$ of twelve requirement variants (listed in Table 3.5) that satisfy the top-level requirement, computed by activating or disabling the optional requirements *NS* and *SJ*, and by making choices for the XOR-refined requirements *NSD* and *SJ*. Fig. 3.5 and Fig. 3.6 report, as an example, a graphical representation of variants $\mathcal{V}_3$ and $\mathcal{V}_4$.

Table 3.5: *The 12 requirement variants of the smart traffic scenario.*

Var.	Description	Requirements
$\mathcal{V}_1$	Static navigation system and static traffic lights	*NS*, *NSD*, *SNS*, *RS*, *J*, *SJ*, *STL*, *TR*
$\mathcal{V}_2$	Adaptive navigation system and static traffic lights	*NS*, *NSD*, *ANS*, *RS*, *J*, *SJ*, *STL*, *TR*
$\mathcal{V}_3$	Only static traffic lights	*J*, *SJ*, *STL*, *TR*
$\mathcal{V}_4$	Only priority lanes signs	*J*, *SJ*, *P*, *TR*
$\mathcal{V}_5$	All panels disabled	*J*, *TR*
$\mathcal{V}_6$	Only static navigation system	*NS*, *NSD*, *SNS*, *RS*, *J*, *TR*
$\mathcal{V}_7$	Only adaptive navigation system	*NS*, *NSD*, *ANS*, *RS*, *J*, *TR*
$\mathcal{V}_8$	Static navigation system and adaptive traffic lights	*NS*, *NSD*, *SNS*, *RS*, *J*, *SJ*, *ATL*, *TR*
$\mathcal{V}_9$	Adaptive navigation system and adaptive traffic lights	*NS*, *NSD*, *ANS*, *RS*, *J*, *SJ*, *ATL*, *TR*
$\mathcal{V}_{10}$	Only adaptive traffic lights	*J*, *SJ*, *ATL*, *TR*
$\mathcal{V}_{11}$	Static navigation system and priority lanes signs	*NS*, *NSD*, *SNS*, *RS*, *J*, *SJ*, *P*, *TR*
$\mathcal{V}_{12}$	Adaptive navigation system and priority lanes signs	*NS*, *NSD*, *ANS*, *RS*, *J*, *SJ*, *P*, *TR*

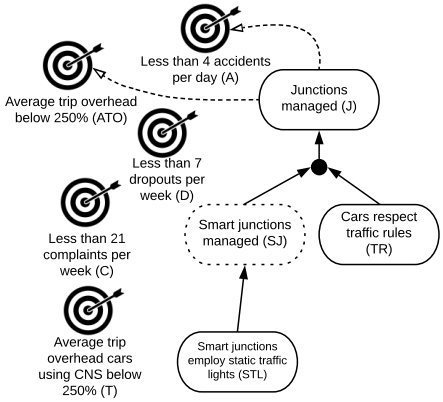


Figure 3.5: *A graphical representation of the requirement variant $\mathcal{V}_3$.*

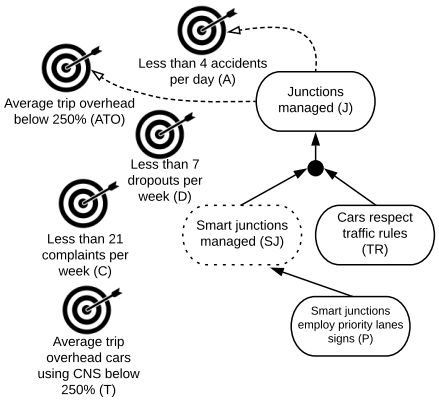


Figure 3.6: *A graphical representation of the requirement variant $\mathcal{V}_4$.*

Definition 6 (system configuration)**.** *Given the set of requirement variants $\mathcal{V}$ and the set of operating contexts $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, a system configuration assigns a requirement variant to each operating context. Formally, a system configuration is a set of pairs $\{\langle \mathcal{C}_1, \mathcal{V}_i \rangle, \ldots, \langle \mathcal{C}_j, \mathcal{V}_k \rangle, \ldots, \langle \mathcal{C}_m, \mathcal{V}_p \rangle\}$ such that $\mathcal{V}_i, \mathcal{V}_k, \ldots, \mathcal{V}_p \in \mathcal{V}$.*

Given the four possible contexts *day-normal*, *day-extreme*, *night-normal*, *night-extreme*, and given the set $\mathcal{V}$ of possible requirement variants, an example of system configuration is $\{\langle$*day-normal*, $\mathcal{V}_3\rangle, \langle$*day-extreme*, $\mathcal{V}_4\rangle, \langle$*night-normal*, $\mathcal{V}_5\rangle, \langle$*night-extreme*, $\mathcal{V}_{10}\rangle\}$.

A certain requirement R is said to be *active* in a context $\mathcal{C}_i$ if $\langle \mathcal{C}_i, \mathcal{V}_j \rangle$ is in the system configuration and $R \in \mathcal{V}_j$. Otherwise R is said *disabled*.

The concepts of requirement variant and system configuration are essential for us to define the notions of *requirement revision*, which is the basic action that the STS Supervisor performs when adapting the STS, on the basis of the learned run-time information concerning assumptions validity.

Definition 7 (requirement revision)**.** *Given a requirements model $\mathcal{RM}$, and a requirement variant $\mathcal{V}_i$ of $\mathcal{RM}$, a revision of a requirement R with respect to $\mathcal{V}_i$ is an operation that returns a different variant $\mathcal{V}_j$ of $\mathcal{RM}$ with $i \neq j$. We distinguish the following types of revisions of a requirement R:*

- *Disabling $R \in \mathcal{V}_i$ returns a $\mathcal{V}_j$ that does not contain R* [2]*.*
- *Activating $R \notin \mathcal{V}_i$ returns a $\mathcal{V}_j$ that contains R.*
- *Relaxing $R \in \mathcal{V}_i$ returns a $\mathcal{V}_j$ such that, given the set $\mathcal{D}$ of descendants of R in $\mathcal{V}_i$, the set of descendants of R in $\mathcal{V}_j$ is $\mathcal{D}' \subset \mathcal{D}$.*
- *Strengthening $R \in \mathcal{V}_i$ returns a $\mathcal{V}_j$ such that, given the set $\mathcal{D}$ of descendants of R in $\mathcal{V}_i$, the set of descendants of R in $\mathcal{V}_j$ is $\mathcal{D}' \supset \mathcal{D}$.*

[2] Clause 3 of Def. 5 ensures that all the descendants of R in $\mathcal{RM}$ are also not in $\mathcal{V}_j$

- *Altering $R \in \mathcal{V}_i$ return a $\mathcal{V}_j$ such that, given the set $\mathcal{D}$ of descendants of R in $\mathcal{V}_i$, the set of descendants of R in $\mathcal{V}_j$ is $\mathcal{D}'$ such that $\mathcal{D}' \neq \mathcal{D}$ and $\mathcal{D}' \cap \mathcal{D} \neq \varnothing$.*

Table 3.6 shows the revisions applied to each of the requirements (*NS*, *NSD*, etc.) in the requirements model of Fig. 3.3, in order to obtain the twelve possible requirement variants starting from $\mathcal{V}_1$. For example, the requirement variant $\mathcal{V}_3$ (see Table 3.5) is obtained from $\mathcal{V}_1$ by *disabling* the requirement *NS* and by consequence also all of its descendants. On the other hand, the requirement variant $\mathcal{V}_2$ is obtained from $\mathcal{V}_1$ by *altering NS* by replacing the descendant requirement *SNS* with *ANS*. In requirement variant $\mathcal{V}_5$, the requirement J is *relaxed* w.r.t. variant $\mathcal{V}_1$, for its descendants *SJ* and *STL* are in $\mathcal{V}_1$ but not in $\mathcal{V}_5$. Notice that if we had started from $\mathcal{V}_5$ instead (not shown in the table), the same requirement J would have been *strengthened* in variant $\mathcal{V}_1$. Finally, requirement P is *activated* in $\mathcal{V}_4$, for it was not present in $\mathcal{V}_1$.

Table 3.6: *Revisions of the requirements in Fig. 3.3 that are performed from requirement variant $\mathcal{V}_1$ to the other eleven requirement variants. The requirements whose nodes are in $\mathcal{V}_1$ are underlined. The last row describes the variant revision type from $\mathcal{V}_1$ to the other requirement variants. Value "-" indicates that no revision is applied.*

	$\mathcal{V}_1$	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$	$\mathcal{V}_6$	$\mathcal{V}_7$	$\mathcal{V}_8$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$
NS	-	alt	dis	dis	dis	-	alt	-	alt	dis	-	alt
NSD	-	alt	dis	dis	dis	-	alt	-	alt	dis	-	alt
SNS	-	dis	dis	dis	dis	-	dis	-	dis	dis	-	dis
ANS	-	act	-	-	-	-	act	-	act	-	-	act
RS	-	-	dis	dis	dis	-	-	-	-	dis	-	-
J	-	-	-	alt	rel	rel	rel	alt	alt	alt	alt	alt
SJ	-	-	-	alt	dis	dis	dis	alt	alt	alt	alt	alt
STL	-	-	-	dis	dis	dis	dis	dis	dis	dis	dis	dis
ATL	-	-	-	-	-	-	-	act	act	act	-	-
P	-	-	-	act	-	-	-	-	-	-	act	act
TR	-	-	-	-	-	-	-	-	-	-	-	-
$\mathcal{V}_1$	-	alt	rel	alt	rel	rel	alt	alt	alt	alt	alt	alt

Def. 8 lifts the notion of revision from an individual requirement (Def. 7) to an entire requirement variant.

Definition 8 (requirement variant revision)**.** *Given a requirements model $\mathcal{RM}$, and given two requirement variants $\mathcal{V}_i, \mathcal{V}_j$ of $\mathcal{RM}$, $\mathcal{V}_j$ is a revision of $\mathcal{V}_i$ if and only if $\mathcal{V}_j \neq \mathcal{V}_i$. A requirement variant revision can be of three types:*

- *Relaxation: for each requirement R in $\mathcal{RM}$, either R is not revised between $\mathcal{V}_i$ and $\mathcal{V}_j$, or it is relaxed or disabled;*
- *Strengthening: for each requirement R in $\mathcal{RM}$, either R is not revised between $\mathcal{V}_i$ and $\mathcal{V}_j$, or it is strengthened or activated;*
- *Alteration: when $\mathcal{V}_j$ is neither a relaxation or a strengthening of $\mathcal{V}_i$.*

The last line of Table 3.6 determines the type of variant revision based on the individual requirements revisions. For example, $\mathcal{V}_3$ is a relaxation of $\mathcal{V}_1$, for the only requirement revision type that is applied is disabling (*NS*, *NSD*, *SNS*, and *RS*). $\mathcal{V}_2$ is instead an alteration of $\mathcal{V}_1$, for the applied requirement revisions do not define neither a relaxation nor a strengthening.

3.6.2 The STS Supervisor Control Loop

The notions introduced in Def. 5–8 are used to explain the *STS Supervisor* (first mentioned in Fig. 3.1) that guides the adaptation of an STS. The control loop of the STS Supervisor is shown in Fig. 3.7 and described in the following.

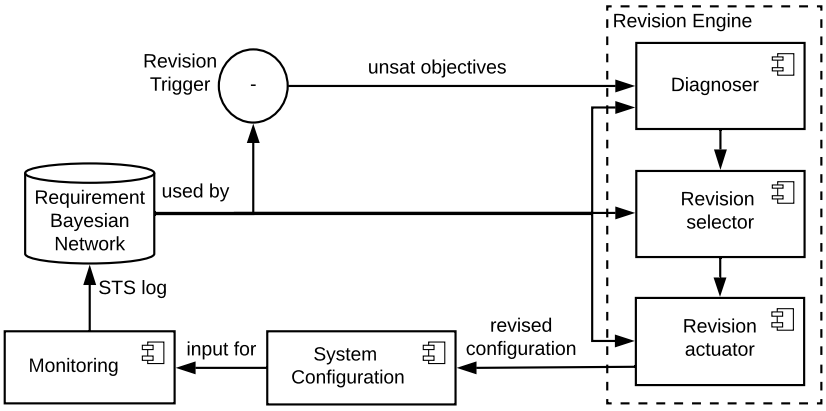


Figure 3.7: *The main components of the STS Supervisor.*

At design-time, an initial system configuration (as per Def. 6) is selected by the analyst according to the available domain knowledge, and it is stored in the *System Configuration* component.

At run-time, the *Monitoring* component collects information about the satisfaction or violation of the requirements and about the operating contexts in which they are evaluated. The system's objectives are also evaluated, typically with lower frequency and relying on aggregate information. This knowledge (the *STS log*) is used to learn, by means of a *Requirement Bayesian Network* (described in Sec. 3.2.2), correlations between the satisfaction of the requirements and the achievement of the objectives in the different contexts.

A *Revision Trigger* component (Sec. 3.6.2) uses the learned knowledge to determine whether some requirements should be revised. The requirements revision process is executed by the *Revision Engine* component that generates as output a (possibly) new system configuration, replacing the current one in the *System Configuration* component. The sub-components of the *Revision Engine* are detailed in Sec. 3.6.2–3.6.2.

The STS Supervisor control loop implements a variant of the hill climbing optimization algorithm. A system configuration is treated as a solution in the space of all possible solutions. We say that the hill climbing optimization process (Supervisor's

control loop) performs a *step* every time a requirement revision process is triggered by the *Revision Trigger*. A new solution (system configuration) is then selected among the solutions in the neighborhood of the current system configuration. The neighborhood of a system configuration is defined by our *Revision Engine* component by making use of the requirements model's structure and of the *RBN*. In particular, in Sec. 3.6.2 we describe two different algorithms for the selection of a requirements' revision that can be used as informed heuristics for the definition of a neighborhood of a system configuration. In Sec. 3.7 we will then evaluate such heuristics by comparing them with uninformed ones that do not leverage run-time execution data about the validity of the assumptions underlying a requirements model.

Revision Trigger

The Revision Trigger determines if a requirements revision is necessary. If so, the *Diagnoser* (Sec. 3.6.2) is invoked; otherwise, no revision of requirements is triggered.

Let e be an event representing network stability: changes in the probability distributions in the *Requirement Bayesian Network* are not significant anymore (i.e., the variations in the distribution when a new sample is given are below a specified threshold). Assuming a consistent behavior of the system, such event will occur after some time.

Let t_{oa} be a threshold defining the minimum objectives achievement joint probability (i.e., the probability oa that all the objectives are achieved together) desired by the system designer. A revision (i.e., a new step of the hill climbing procedure) is triggered every time e occurs and t_{oa} is not met with the current system configuration (i.e., $oa < t_{oa}$ with the current system configuration). For example, $oa \geqslant 0.95$ indicates a threshold t_{oa} of 95% for the objectives achievement joint probability.

Revisions are triggered based on an analysis of the *objective achievement assumptions*. The revision trigger calculates the joint degree of validity of the objective achievement assumption for all the objectives. A revision is triggered when such value is below $t_{oa} - (1 - t_{oa})$.

Please note that, as described above, at any time instant a certain system configuration $\mathcal{C}_i$ is chosen. The objectives achievement joint probability oa, therefore, depends on the chosen configuration. For instance, for the running example, if $\mathcal{C}_i = \{\langle$*day-normal*, $\mathcal{V}_3\rangle, \langle$*day-extreme*, $\mathcal{V}_4\rangle, \langle$*night-normal*, $\mathcal{V}_5\rangle, \langle$*night-extreme*, $\mathcal{V}_{10}\rangle\}$, then oa needs to be calculated as follows:

$$\begin{aligned} oa = P(\mathbf{O}_{true} |\ \mathbf{dn}\ \wedge\ \mathbf{v3})P(\mathbf{dn}) + P(\mathbf{O}_{true} |\ \mathbf{de}\ \wedge\ \mathbf{v4})P(\mathbf{de}) + \\ P(\mathbf{O}_{true} |\ \mathbf{nn}\ \wedge\ \mathbf{v5})P(\mathbf{nn}) + P(\mathbf{O}_{true} |\ \mathbf{ne}\ \wedge\ \mathbf{v10})P(\mathbf{ne}) \end{aligned} \tag{3.10}$$

with $\mathbf{dn}, \mathbf{de}, \mathbf{nn}, \mathbf{ne}$ evidences for the contexts *day-normal*, *day-extreme*, *night-normal*, *night-extreme*, respectively (e.g., $\mathbf{dn} = (Time_{day} \wedge Weather_{normal})$), and $\mathbf{v3}, \mathbf{v4}, \mathbf{v5}, \mathbf{v10}$ evidences for the values of the requirements in the requirement variants $\mathcal{V}_3$, $\mathcal{V}_4$, $\mathcal{V}_5$, $\mathcal{V}_{10}$, respectively (e.g., $\mathbf{v3} = (J_{act} \wedge SJ_{act} \wedge STL_{act} \wedge TR_{act} \wedge \mathbf{V3D}_{dis})$, where $\mathbf{V3D}$ is the set of remaining requirement nodes disabled in $\mathcal{V}_3$).

Analogously, any probability that needs to be calculated on the *Requirement Bayesian Network* w.r.t. a certain context should take into account the currently chosen system configuration. In order to ease the reading, however, in the rest of the

chapter we do not explicitly represent (unless differently specified) the evidence for the requirement nodes. We implicitly assume, instead, that in a certain context **c** the given evidence informs also about the active/disabled requirements in the requirement variant currently chosen for context **c**. For instance, if $\mathcal{V}_3$ is currently chosen for context **c**, and we want to calculate the objectives achievement joint probability in such context **c**, instead of writing $P(\mathbf{O}_{true}|\ \mathbf{c} \wedge \mathbf{v3})$ (with **v3** defined as above) we simply write $P(\mathbf{O}_{true}|\ \mathbf{c})$.

Diagnoser

When a revision is triggered, the *Diagnoser* component is invoked to determine the reasons why the objectives are not achieved. To do so, it uses the *Requirement Bayesian Network* to determine the most problematic operating context in which the objectives are not achieved.

Let **all** be the set of all possible assignments of a value to each of the context variables in the Bayesian Network (e.g., for the Bayesian Network reported in Fig. 3.4, **all** = $\{\{Time_{day}, Weather_{normal}\}, ..., \{Time_{night}, Weather_{extreme}\}\}$). The most problematic context (denoted with **mpc**) is the assignment resulting from Equation 3.11.

$$\mathbf{mpc} = argmax_{\mathbf{c} \in \mathbf{all}} P(\mathbf{O}_{false} \mid \mathbf{c}) \tag{3.11}$$

Revision Selector

Let $\mathcal{V}_{\mathbf{mpc}}$ be the requirement variant assigned to the most problematic context **mpc** in the current system configuration. The *Revision Selector* determines the most adequate requirements revisions to perform to requirements in $\mathcal{V}_{\mathbf{mpc}}$ so to increase $P(\mathbf{O}_{true} \mid \mathbf{mpc})$. Our framework includes two heuristic algorithms: pureBN (PB) and stateBased (SB).

PB and SB first identify the relationship between the requirement and the objective nodes in the Bayesian Network by performing an analysis of some of the design-time assumptions described in Sec. 3.5.

Requirements can be either *useful* for the achievement of the system's objectives or *harmful*. Useful requirements can be further divided into requirements that are *more useful when obeyed* and requirements that are *more useful when violated*. Useful requirements can also be either *often obeyed* when the objectives are not achieved or *often violated*.

Let us formalize this classification in terms of probability theory. Let **R** be the set of all requirement nodes in a *Requirement Bayesian Network*. In the rest of this section, for simplicity, let $\mathbf{R} = \{X, Y, Z\}$.

Harmful requirements. The set of requirements such that, when all disabled, guarantee a better objectives achievement joint probability than when at least one of them is activated. Let **da** be the set of all possible assignments of values in the set $\{dis, act\}$ to all nodes **R** (e.g., given $\mathbf{R} = \{X, Y, Z\}$, then **da** = $\{\{X_{dis}, Y_{dis}, Z_{dis}\}, \ldots, \{X_{act}, Y_{act}, Y_{act}\}\}$).

Let $\mathbf{h}$ be the assignment of Equation 3.12 (e.g., $\mathbf{h} = \{X_{dis}, Y_{act}, Z_{act}\}$).

$$\mathbf{h} = argmax_{\mathbf{r} \in \mathbf{da}}\ P(\mathbf{O}_{true} | \mathbf{r}\ \wedge \mathbf{mpc}) \tag{3.12}$$

Let $\mathbf{D} \subseteq \mathbf{R}$ be the set of nodes that have value *dis* in $\mathbf{h}$, and $\mathbf{A}$ be the set $\mathbf{R} \backslash \mathbf{D}$ (e.g., $\mathbf{D} = \{X\}$ and $\mathbf{A} = \{Y, Z\}$). Harmful requirements are all the requirements such that the corresponding nodes in the Bayesian Network are in $\mathbf{D}$. Useful requirements are instead all the requirements such that the corresponding nodes in the Bayesian Network are in $\mathbf{A}$.

Note that an harmful (useful) requirement is one whose associated *requirement necessity assumption* is negative (positive).

Requirements that are more useful when obeyed (violated). The set of requirements that are most useful for the objectives achievement joint probability when active, either when obeyed or violated.

Let $\mathbf{ov}$ be the set of all possible assignments of values in the set $\{ob, viol\}$ to all nodes in the set of useful requirements $\mathbf{A}$ (e.g., given $\mathbf{A} = \{Y, Z\}$, then $\mathbf{ov} = \{\{Y_{ob}, Z_{ob}\}, \{Y_{ob}, Z_{viol}\}, \{Y_{viol}, Z_{ob}\}, \{Y_{viol}, Z_{viol}\}\}$). Let $\mathbf{u}$ be the assignment resulting from Equation 3.13 (e.g., $\mathbf{u} = \{Y_{ob}, Z_{viol}\}$).

$$\mathbf{u} = argmax_{\mathbf{r} \in \mathbf{ov}}\ P(\mathbf{O}_{true} | \mathbf{r}\ \wedge \mathbf{mpc} \wedge \mathbf{D}_{dis}) \tag{3.13}$$

Requirements that are more useful when obeyed (violated) are all the requirements whose nodes in the Bayesian Network have value *ob* (*viol*) in $\mathbf{u}$ (e.g., Y is more useful when obeyed, while Z is more useful when violated).

Determining whether a requirement is useful when obeyed or when violated corresponds to evaluate if the associated *contribution assumption* is positive or negative. When a requirement has no *aims at* link to an objective in $\mathcal{RM}$, we are evaluating a hypothetical link between the two elements.

Useful requirements often obeyed (violated) when $\mathbf{O}_{false}$. The set of useful requirements that are most likely to be obeyed (violated) when the objectives are not achieved. Let $\mathbf{mle}$ be the assignment of Equation 3.14 (e.g., $\mathbf{mle} = \{Y_{ob}, Z_{ob}\}$). We call such assignment *most likely explanation* for $\mathbf{O}_{false}$ in $\mathbf{mpc}$.

$$\mathbf{mle} = argmax_{\mathbf{r} \in \mathbf{ov}}\ P(\mathbf{r}\ | \mathbf{O}_{false} \wedge \mathbf{mpc} \wedge \mathbf{D}_{dis}) \tag{3.14}$$

Useful requirements that are often obeyed (violated) when $\mathbf{O}_{false}$ are those whose corresponding nodes in the *Requirement Bayesian Network* have value *ob* (*viol*) in $\mathbf{mle}$ (e.g., both nodes Y and Z are often obeyed when $\mathbf{O}_{false}$).

The most likely explanation $\mathbf{mle}$ for $\mathbf{O}_{false}$ in $\mathbf{mpc}$ is determined by computing the most likely degree of validity of the *requirement satisfiability assumptions* of the requirements that are not harmful. The most likely value of a requirement R is *obeyed* if the most likely degree of validity $\delta_S(R, (\mathbf{mpc}, \mathbf{O}_{false}))$ is positive, otherwise the most likely value is *violated*.

Algorithm PB. After identifying the relationship between requirements and system's objectives, as just described, this algorithm applies the following procedure, also illustrated by the decision tree of Fig. 3.8:

1. *Disable/Relax* harmful requirements.
2. *Relax* useful requirements that are more useful when violated.
3. *Strengthen/Alter* useful requirements that are more useful when obeyed but they are often violated when $\mathbf{O}_{false}$.
4. Keep all other requirements unrevised, or *strengthen* them.

For example, given $\mathbf{R}$ as described above, PB suggests to disable/relax X, to relax Z and to either leave unaltered Y or to strengthen it.

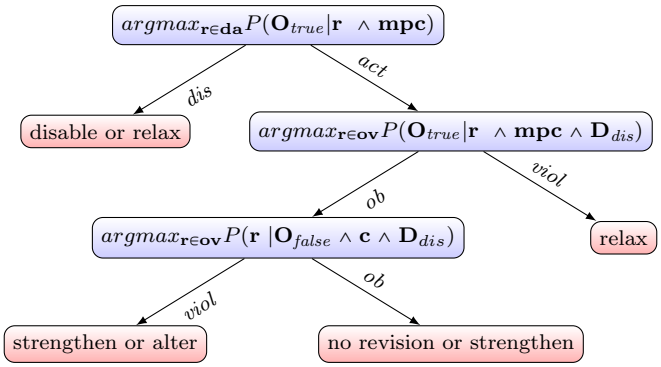


Figure 3.8: *Decision tree used by algorithm* PB *for determining a suitable type of revision.*

Algorithm 2 reports the pseudocode for PB. Line 2 determines, based on the top

Algorithm 2 The PB algorithm for revision selection

```
1: function PB(mpc)
2:     R' ← getBestReq(mpc)                          ▷ obtain A' and D'
3:     if (|A| > 0) && ¬allTried(mpc) then
4:         mle ← getMLE(O_false, mpc)
5:         for all reqs R ∈ A' do
6:             u ← getUsefulVal(R, mpc)
7:             setSugg({R}, getSugg(R, mle, u))
       return R'
```

decision node of Fig. 3.8, the set of requirements $\mathbf{R}'$ that has the highest probability to satisfy the objectives. If the current requirement variant $\mathbf{R}$ has some active requirements, and not all requirement variants have been attempted in the most problematic context yet (line 3), the most likely explanation **mle** for not fulfilling the objectives is determined (line 4). Then, for each active requirement in the new set of requirements

$\mathbf{R}'$ (line 5), the algorithm determines the "useful" state, i.e., whether requirements is more useful when obeyed or violated (line 6). Finally, in line 7, the suggestion for the examined requirement is determined based on the decision tree of Fig. 3.8. If no requirements are active or all possible variants have already been tried, PB returns $\mathbf{R}'$: the best possible requirement variant for the most problematic context (skipping lines 4–7).

Algorithm SB. This algorithm implements a different strategy for the revision selection, for it analyzes the relationship between the *average* requirements satisfaction (calculated as the mean) and the objectives achievement joint probability of the current system configuration. Fig. 3.9 plots five examples of system configurations in four states with respect to the average requirements satisfaction and the objectives achievement joint probability.

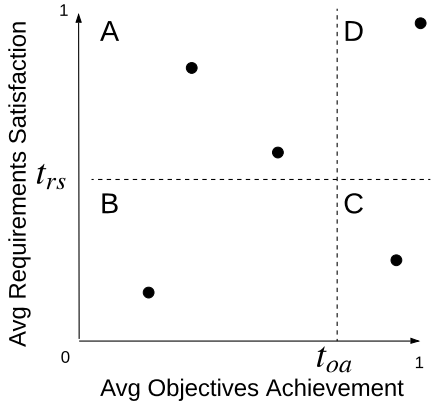


Figure 3.9: *Plotting system configurations (points) in four states (A–D) according to the average requirements satisfaction and objectives achievement joint probability.* t_{rs} *and* t_{oa} *denote the desired average requirements satisfaction and objectives achievement joint probabilities, respectively.*

Configurations in state A sufficiently satisfy the requirements, but this does not lead to sufficient objectives achievement. State B has insufficient requirements satisfaction and objectives achievement. State C indicates that the objectives are achieved even though the requirements are not satisfied. State D is the ideal area: the requirements are satisfied and the objectives are achieved. Algorithm SB aims to revise the system configuration and move the system into state D by applying the following procedure:

1. Calculate average requirements satisfaction probability.
2. Calculate objectives achievement joint probability.
3. *Disable* harmful requirements, if any. Else, go to point 4.
4. If the system configuration is in state A: *Relax* useful requirements that are more useful when violated but often obeyed when $\mathbf{O}_{false}$, if any. Otherwise, *Strengthen*/*Alter* all useful requirements.

5. If the system configuration is in state B: *Strengthen/Alter* useful requirements that are more useful when obeyed but often violated when $\mathbf{O}_{false}$ and *Relax* useful requirements that are more useful when violated.

6. If the system configuration is in state C: *Relax* useful requirements that are more useful when violated and often violated when $\mathbf{O}_{false}$, if any. Otherwise, *Strengthen/Alter* useful requirements that are more useful when obeyed but often violated when $\mathbf{O}_{false}$.

For example, given $\mathbf{R}$ as described above, SB only suggests to disable X, for such requirement is *harmful*.

Algorithm 3, reports the pseudocode for SB. SB first determines the average requirements satisfaction and objectives achievement based on the evidence from the active requirements in the most problematic context (lines 2). Like in PB, the best possible requirement variant in the most problematic context is generated (line 3). If there are currently no active requirements or $\mathbf{R}'$ contains at least one suggestion of type `disable`, the function returns immediately (line 4).

Three empty sets of requirements are defined in line 5: requirements that are often obeyed but they are better (i.e., useful) when violated *obbv*, requirements that are often violated and they are also better when violated *vabv*, and requirements that are often violated but they are better when obeyed *vbbo*. After determining

Algorithm 3 The SB algorithm for revision selection

```
1:  function SB(mpc)
2:      rs ← avgReqSat(A); oa ← avgObjAch(A)
3:      R′ ← getBestReq(mpc)                          ▷ obtain A′ and D′
4:      if (|A| = 0) || hasDis(R′)) then return R′
5:      obbv ← vabv ← vbbo ← {}
6:      mle ← getMLE(O_false, mpc)
7:      u ← getUsefulVal(A′, mpc)
8:      determineTYPE(A′, mle, u, obbv, vabv, vbbo)
9:      if rs ⩾ t_rs && oa < t_oa then
10:         if |obbv| > 0 then setSugg(obbv, relax)
11:         else setSugg(A, alter ∨ strengthen)
12:     else if rs < t_rs && oa < t_oa then
13:         setSugg(oobv ∪ vabv, relax); setSugg(vbbo, alter ∨ strengthen)
14:     else if rs < t_rs && oa ⩾ t_oa then
15:         if |vabv| > 0 then setSugg(vabv, relax)
16:         else if |vbbo| > 0 then setSugg(vbbo, alter ∨ strengthen)
        return R′
```

the most likely explanation **mle** (line 6), the algorithm determines (line 7) for all active requirements of the new requirement variant whether they are more useful when obeyed or violated. Using **mle** (obeyed / violated) and the useful value (obeyed / violated), the requirements are added to the corresponding sets *obbv*, *vavb*, and *vbbo* (line 8).

Lines 9–16 compare the state of the current requirement variant (rs and oa) against the thresholds t_{rs} and t_{oa}. If the requirement variant is in state A (lines 9–11), if *obbv* contains requirements, the suggestion is to relax them; if *obbv* is empty, a suggestion to alter or strengthen is given for the active requirements (the active requirements behave as expected but the objectives are not achieved). In state B (lines 12-13), a relaxation of the requirements that are useful when violated is suggested, and an alteration or strengthening is suggested for the requirements that are useful if obeyed. In state C (lines 14–16), if there are violated requirements that are better if violated, it is suggested that they are relaxed; if, instead, there are violated requirements that are better if obeyed, the suggestion is to either alter or strengthen them.

PB and SB adopt different strategies for the revision of requirements. While PB determines for all the requirements the most opportune revision to perform (if any), SB considers the global state of the system and suggests to revise only a certain type of requirements at every iteration. This difference leads to a different definition of the neighborhoods of the configurations during the hill climbing process (see Sec. 3.6.2), and this leads to different results (as will be visible in Sec. 3.7).

Revision Actuator

This component adopts a new requirement variant in the **mpc**. Given a list of suggested revisions for requirements in the requirement variant $\mathcal{V}_{\mathbf{mpc}}$ currently assigned to context **mpc** in the system configuration, the Revision Actuator selects a requirement variant $\mathcal{V}_j$ that is as much aligned as possible with the direction provided by the suggestion.

For example, consider $\mathcal{V}_1$, and assume the Revision Selector suggests to alter the requirement *NS*. Then, the Revision Actuator has to find other requirement variants where *NS* is altered from $\mathcal{V}_1$ (e.g., $\mathcal{V}_2$, $\mathcal{V}_7$, $\mathcal{V}_9$ or $\mathcal{V}_{12}$, see Table 3.6). The obtained set of variants defines the neighborhood of the current system configuration in the hill climbing optimization process.

If the neighborhood contains multiple variants, different distance metrics can be defined, e.g., the similarity with the current variant, or the sensitivity of the objectives to the change of the selected requirements. Here, we adopt the number of revisions of requirements needed to obtain $\mathcal{V}_j$ from $\mathcal{V}_{\mathbf{mpc}}$. For instance, four revisions are necessary to obtain $\mathcal{V}_2$ from $\mathcal{V}_1$, seven revisions are necessary to obtain $\mathcal{V}_7$ from $\mathcal{V}_1$, etc.

After selecting the new requirement variant $\mathcal{V}_j$, the current system configuration is updated to map the context **mpc** to the new $\mathcal{V}_j$ instead of $\mathcal{V}_{\mathbf{mpc}}$.

If there is no new variant that is aligned with the provided suggestion (i.e., the neighborhood is empty or it contains only already-attempted variants), the Revision Actuator randomly selects a system configuration never tried before, if any. This makes our implementation of hill climbing different from traditional ones, and guarantees convergence to an optimal solution.

3.7 Evaluation

We report on an evaluation of the proposed supervision mechanism; in particular, we conduct an experiment that investigates the process through which the Supervisor's control loop identifies an optimal system configuration.

3.7.1 Scope, Context, and Hypotheses

The object of our study consists of the requirement revision heuristics. We compare two sets of dependent variables:

i. *Informed heuristics*: the two algorithms PB and SB described in Sec. 3.6.2 implemented in our *Revision Engine*; and

ii. *Uninformed heuristics*: three baseline algorithms that do not leverage knowledge about the validity of the design-time assumptions:

   1. Maximum distance 8 (D8) defines a neighbourhood composed of all the system configurations that are obtained by revising at most 8 requirements[3];
   2. Maximum size 10 (S10) defines a neighborhood composed of the 10 closest system configurations to the current one;
   3. Maximum size 20 (S20) defines a neighborhood composed of the 20 closest system configurations to the current one.

We identify four independent variables for studying the process through which the Supervisor's control loop identifies an optimal system configuration:

1. *Convergence speed*: the number of steps (i.e., revisions triggered by the *Revision Trigger*, as described in Sec. 3.6.2) and the number of explored system configurations that the Supervisor's control loop requires before it identifies an optimal system configuration;

2. *Quality*: the probability that the system configurations explored satisfy the system's objectives;

3. *Stability*: the number of requirements revisions that are performed while identifying an optimal system configuration.

Furthermore, for the informed algorithms alone, we evaluate *4. Noise tolerance*: the degree to which the amount of noisy input data (imperfect monitors) affects convergence speed, quality, and stability. Noise tolerance does not affect uninformed algorithms, for they do not take into account any information about requirements satisfaction.

Our experiment is run through CrowdNavExt, the simulation environment that instantiates the smart traffic example presented in Sec. 3.3. Within the context of such simulation environment, we formulate the following hypotheses:

[3]The value of 8 was chosen via experimentation with CrowdNav. Revising one requirement leads to a distance of 4–5 from the original system configuration, and each system configuration has 10%–20% of all system configurations in its neighborhood.

- H_1: our informed heuristics provide a higher convergence speed than the uninformed heuristics;
- H_2: our informed heuristics allow to higher-quality system configurations than the uninformed heuristics;
- H_3: our informed heuristics allow to perform less revisions than the uninformed heuristics while finding an optimal system configuration;
- H_4: noisy input data has a marginal effect on convergence speed, quality, and stability of the Supervisor's control loop when using our informed heuristics.

3.7.2 Design and Instrumentation

SASS (Supervisor of Autonomous Software Systems)[4] is our implementation of the Supervisor's control loop described in Sec. 3.6 as a modified version of hill climbing. The supervisor performs a local search and stops when either (i) all the system configurations have been tried; or (ii) a local optimum (system configuration) is found that has objectives achievement joint probability *oa* above the desired threshold t_{oa}. This probability is determined from simulation data (see Sec. 3.6.2) as the joint probability of achievement of all the objectives, given the chosen system configuration. We call optimal the last system configuration chosen, since either it is above the desired threshold or there is no other better configuration.

CrowdNavExt has $12^4 = 20,736$ possible system configurations, i.e., assignments of one of the twelve variants to each of the four contexts (see Def. 6). To keep our simulation time manageable, we chose 81 system configurations via test case generation techniques. We first applied *pairwise testing*: for each pair of variables, we obtained all their possible discrete combinations. Our variables are: time of the day (day, night), weather (normal, extreme), the alternative requirements for the navigation service (none, adaptive, static), and the alternatives for managing smart junctions (none, adaptive lights, static lights, priority lanes). This led to 3 different variants for each of the 4 operating contexts, using pairwise testing. We generated all combinations of the four groups of variants (each system configuration includes four variants, one per each operating context). Finally, we introduced three additional system configurations more distant from the others (in terms of number of required revisions, as described in Sec. 3.6.2). Two of them are the best-scoring system configurations. Therefore, in our experiments, we study 84 system configurations (reported in Table 3.7).

Table 3.8 describes all the simulation parameters of our experiments. We run simulations with three possible values of t_{oa}: 0.35, 0.3, and 0.25. These values have been determined manually, based on the objectives achievement joint probability *oa* of the 84 system configurations (shown in Fig. 3.10), so that the three different values determine, as also reported in Table 3.8, three levels of difficulty for the search of an optimal configuration in terms of percentage of system configurations above the threshold.

[4]SASS' code repository: https://bitbucket.org/dellannadavide/sass.

Table 3.7: *The 84 system configurations employed for the experiments.* ***nn***, ***dn***, ***ne***, ***de*** *respectively represent the contexts night-normal, day-normal, night-extreme and day-extreme.*

conf	**nn**	**dn**	**ne**	**de**	conf	**nn**	**dn**	**ne**	**de**	conf	**nn**	**dn**	**ne**	**de**
1	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	29	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	57	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_5$
2	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	30	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	58	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_8$
3	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	31	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	59	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_1$
4	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	32	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	60	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_5$
5	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	33	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	61	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_8$
6	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	34	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_8$	62	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_1$
7	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_8$	35	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_1$	63	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_5$
8	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_1$	36	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_5$	64	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_8$
9	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_5$	37	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	65	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_1$
10	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	38	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	66	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_5$
11	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	39	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	67	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_8$
12	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	40	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	68	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_1$
13	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	41	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	69	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_5$
14	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	42	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	70	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_8$
15	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	43	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_8$	71	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_1$
16	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_8$	44	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_1$	72	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_5$
17	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_1$	45	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_5$	73	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_8$
18	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_5$	46	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	74	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_1$
19	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	47	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	75	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_5$
20	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	48	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	76	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_8$
21	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	49	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	77	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_1$
22	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	50	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	78	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_5$
23	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	51	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	79	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_8$
24	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	52	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_8$	80	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_1$
25	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_8$	53	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_1$	81	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$
26	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_1$	54	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$	82	$\mathcal{V}_1$	$\mathcal{V}_3$	$\mathcal{V}_5$	$\mathcal{V}_8$
27	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$	55	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	83	$\mathcal{V}_3$	$\mathcal{V}_2$	$\mathcal{V}_5$	$\mathcal{V}_8$
28	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	56	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	84	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_6$	$\mathcal{V}_3$

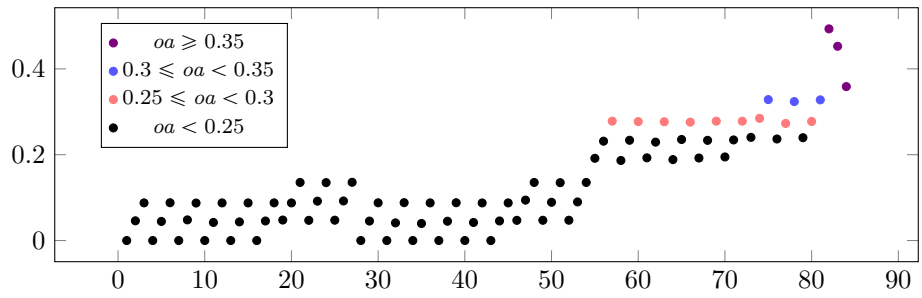


Figure 3.10: *Objectives achievement joint probability (y-axis) for all the 84 system configurations (x-axis).*

We test SASS with the two informed algorithms described in Sec. 3.6 (PB and SB) as heuristics for defining the neighborhood of a system configuration, i.e., the

set of all the other system configurations that satisfy the suggestions provided by the suggestion selector with the help of the trained Bayesian Network. Moreover, we test the three additional uninformed heuristics D8, S20, S20 described above for use as baseline. Note that, in terms of Supervisor's control loop, the uninformed heuristics differ from the informed ones in that they do not use the *Revision Engine* to select a new system configuration. Since the rest of the control loop is common, for the sake of readability, we mention a certain heuristic algorithm (e.g., SB) to refer to the version of the Supervisor's control loop that uses such algorithm (e.g., the Supervisor's control loop with the SB heuristic).

Table 3.8: *Simulation parameters for our experiment with CrowdNavExt*

Parameter	Value	Description
Time	day	600 vehicles in the city
	night	300 vehicles in the city
Weather	normal	speed limits as per CrowdNav
	extreme	speed limits reduced by 25%
Objectives' achievement threshold	0.35	3.5% system configurations above the threshold
	0.3	7% system configurations above the threshold
	0.25	17.8% system configurations above the threshold
Hill climbing heuristic	D8	Uninformed, neighbors max 8 revised revisions
	S10	Uninformed, up to 10 neighbors
	S20	Uninformed, up to 20 neighbors
	PB	Informed, PUREBN
	SB	Informed, STATEBASED

In order to obtain significant data, due to the stochastic nature of the simulation data, SASS has been executed starting from all of the 84 possible system configurations with all the five tested heuristics.

We use the following metrics to determine whether our hypotheses hold:

- *Convergence speed* (H_1, detailed in Sec. 3.7.3)
    1. *Number of steps*: the average number of steps that an algorithm attempts before stopping (i.e., before finding a configuration above the desired threshold t_{oa}).
    2. *Number of explored configurations*: the average percentage of system configurations that an algorithm attempts before stopping.
- *Quality* (H_2, detailed in Sec. 3.7.3)
    1. *Final conf*: the average objectives achievement joint probability (i.e., the average *oa*) of the final solutions determined by an algorithm.
    2. $\overline{oa}_A$: the average *oa* of all the configurations tried by an algorithm A before stopping.

3. $\overline{oa}_{A,last}$: the average *oa* of all the configurations tried by an algorithm A until all algorithms terminate all the 84 executions. Note that, if A terminates before other algorithms, the average will count, for the remaining steps, the *oa* of the last (optimal) configuration found.
4. $\overline{oa}_{A,first}$: the average *oa* of all the configurations tried by an algorithm A until the fastest algorithm terminates all the 84 executions (note that A is not necessary the fastest algorithm).
5. $\overline{oa}_{A,thres}$: the average *oa* of all the configurations tried by an algorithm A until the fastest algorithm reaches the threshold t_{oa} (note that this does not necessary mean that the fastest algorithm has terminated all its executions, nor that algorithm A has reached the threshold).

- *Stability* (H_3, detailed in Sec. 3.7.3)
  1. *Revisions per step*: the average number of requirements revisions performed by an algorithm at each step before to reach the final solution.
  2. *Total revs*: the average total number of revisions performed for a given algorithm to reach the final solution.
- *Noise tolerance* (H_4, only for PB and SB and detailed in Sec. 3.7.4): the variation in performance (speed, quality, stability) when noisy data concerning requirements satisfaction are used to train the *Requirement Bayesian Network*.

In the rest of the section, we present and discuss the results obtained in our experimentation w.r.t. the metrics and hypotheses described above.

3.7.3 Informed vs. uninformed heuristics: speed, quality, and stability

Table 3.9 summarizes the results concerning H_1–H_3 obtained with the five tested heuristics. The table presents the results for the three tested thresholds of t_{oa} and reports the values (average and standard deviation) obtained from the 84 simulation runs for each of the metrics described in the previous section. As stated earlier, the baseline uninformed heuristics are denoted as D8, S10 and S20, while our informed heuristics are denoted as PB and SB.

Convergence speed (H_1)

Number of steps. With all the thresholds, our informed heuristics consistently outperform the uninformed algorithms in terms of number of steps: see the *# steps* column of Table 3.9 and the bar chart of Fig. 3.11, both reporting the average number of steps that each algorithm attempted before stopping.

With $t_{oa} = 0.35$, the three uninformed heuristics D8, S10, S20 take on average 67.8 steps. Our PB and SB heuristics, instead, explore on average only 43.05 system configurations. In this scenario, the few (3 out of 84) optimal system configurations are slightly more distant (in terms of number of necessary revisions) from the non-optimal ones: while the average distance between the 81 system configurations is 16,

Table 3.9: *Comparison of the algorithms with the three objective achievement thresholds. All values are the average over the 84 different simulations. Bold values indicate the best performing algorithm for each metric with each threshold.*

	Speed				Quality										Stability			
A	# steps		% explored configs		Final conf		$\overline{oa}_A$		$\overline{oa}_{A,last}$		$\overline{oa}_{A,first}$		$\overline{oa}_{A,thres}$		Revisions per step		Total revs	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
$t_{oa} = 0.35$																		
D8	75.67	15.79	0.76	0.15	**0.49**	0.02	0.17	0.11	0.22	0.10	0.18	0.03	0.18	0.04	3.84	0.76	301.29	63.95
S10	63.82	15.09	0.68	0.15	**0.49**	0.02	0.16	0.11	0.26	0.13	0.21	0.08	0.17	0.04	3.96	0.79	262.02	64.62
S20	64.05	14.10	0.72	0.16	**0.49**	0.02	0.17	0.11	0.26	0.14	0.21	0.09	0.16	0.05	4.36	0.88	289.24	63.30
PB	48.56	15.74	0.49	0.16	0.49	0.03	**0.18**	0.11	0.32	0.12	0.28	0.10	**0.20**	0.02	3.28	0.74	168.90	66.66
SB	**37.63**	17.19	**0.42**	0.18	**0.49**	0.02	0.17	0.11	**0.36**	0.13	**0.33**	0.12	0.20	0.06	**2.41**	0.56	**97.68**	53.47
$t_{oa} = 0.3$																		
D8	23.77	10.91	0.27	0.12	0.33	0.02	0.16	0.10	0.26	0.08	0.18	0.06	0.17	0.05	3.70	1.08	94.98	45.26
S10	30.01	15.40	0.34	0.16	0.33	0.02	0.16	0.10	0.24	0.07	0.18	0.05	0.17	0.05	3.84	1.15	123.80	62.85
S20	37.02	17.26	0.43	0.19	0.33	0.03	0.15	0.11	0.22	0.07	0.14	0.04	0.13	0.02	4.14	1.20	165.58	79.44
PB	14.25	9.79	0.16	0.10	**0.34**	0.04	**0.18**	0.11	0.29	0.05	**0.24**	0.04	**0.23**	0.03	3.02	0.98	48.06	37.82
SB	**12.56**	8.31	**0.15**	0.09	0.34	0.05	0.16	0.11	**0.30**	0.06	0.23	0.06	0.21	0.04	**2.78**	1.58	**28.36**	15.54
$t_{oa} = 0.25$																		
D8	13.69	8.81	0.16	0.10	0.30	0.04	0.14	0.11	0.24	0.07	0.13	0.02	0.13	0.02	3.43	1.68	56.15	34.77
S10	13.15	9.10	0.15	0.09	0.29	0.04	0.13	0.10	0.23	0.06	0.13	0.02	0.13	0.02	3.48	1.73	54.15	34.75
S20	18.43	9.65	0.22	0.11	0.30	0.04	0.12	0.10	0.21	0.08	0.13	0.02	0.13	0.02	3.85	1.86	85.93	45.28
PB	5.62	6.08	0.08	0.07	**0.29**	0.03	0.14	0.10	0.27	0.04	0.20	0.03	0.17	0.03	**3.07**	1.72	24.49	30.31
SB	**1.77**	1.44	**0.03**	0.02	**0.29**	0.03	**0.18**	0.11	**0.28**	0.03	**0.25**	0.05	**0.20**	0.06	4.60	2.61	**9.05**	6.15

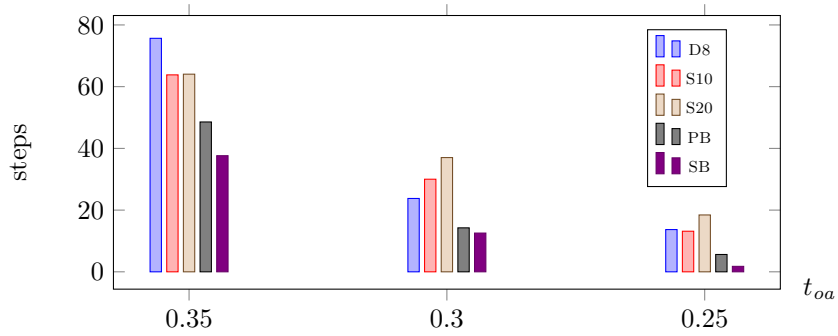


Figure 3.11: *Average number of steps (y-axis) required to find an optimal solution for each of the 3 tested thresholds (x-axis).*

that with the remaining 3 is 20. This affects the number of steps required to find one of them, for all the algorithms give priority to the closest system configurations in the neighborhood. Despite this difficulty, however, PB and SB deliver an improvements of $36.5\% = 1 - (43.05/67.8)$ over the uninformed heuristics in terms of required steps.

With $t_{oa} = 0.3$, the improvement over uninformed heuristics is even higher: $55.7\% = 1 - (13.4/30.27)$, and the efficiency gain increases further with $t_{oa} = 0.25$: $75.5\% = 1 - (3.69/15.09)$. Notably, with this last threshold, SB requires on average only 1.77 steps, i.e., it finds an optimal system configuration after only one or two revisions.

In Fig.3.12, we show the percentage of steps required by the algorithms in order to terminate the first, second and third quartiles (respectively 25, 50 and 75%) of the 84 execution and the first 95% of them. It is worth noting that, in the case of $t_{oa} = 0.35$ (and similarly for the other thresholds), SB terminates the 95% of all the executions before any other uninformed heuristic terminates the first quartile.

When we compare our two informed heuristics, SB outperforms PB. PB suggests different revision types for different requirements. The selection of a requirement variant that satisfies the given suggestions, however, depends on the number of available variants (only 12 in our working example). The suggestions of SB affect, instead, requirements in the same quadrant of Fig. 3.9, thereby moving the current system configuration step-by-step toward the high requirements satisfaction and high objective achievement area. This strategy, which in almost all cases proved to be very efficient, may however result less appropriate when bigger variations in the full set of requirements are needed. For instance, Fig.3.12 shows that, in case of $t_{oa} = 0.35$, SB could not find the optimal solution for two particular executions without trying almost all the possible system configurations (see SB in Fig.3.12a between 0.95 and 1).

Configurations exploration. In terms of the percentage of explored system configurations (reported in the *% explored config* column of Table 3.9), the results show that the informed strategies explore a smaller portion of the possible system configurations than the informed strategies. Notably, in case of $t_{oa} = 0.35$, SB results

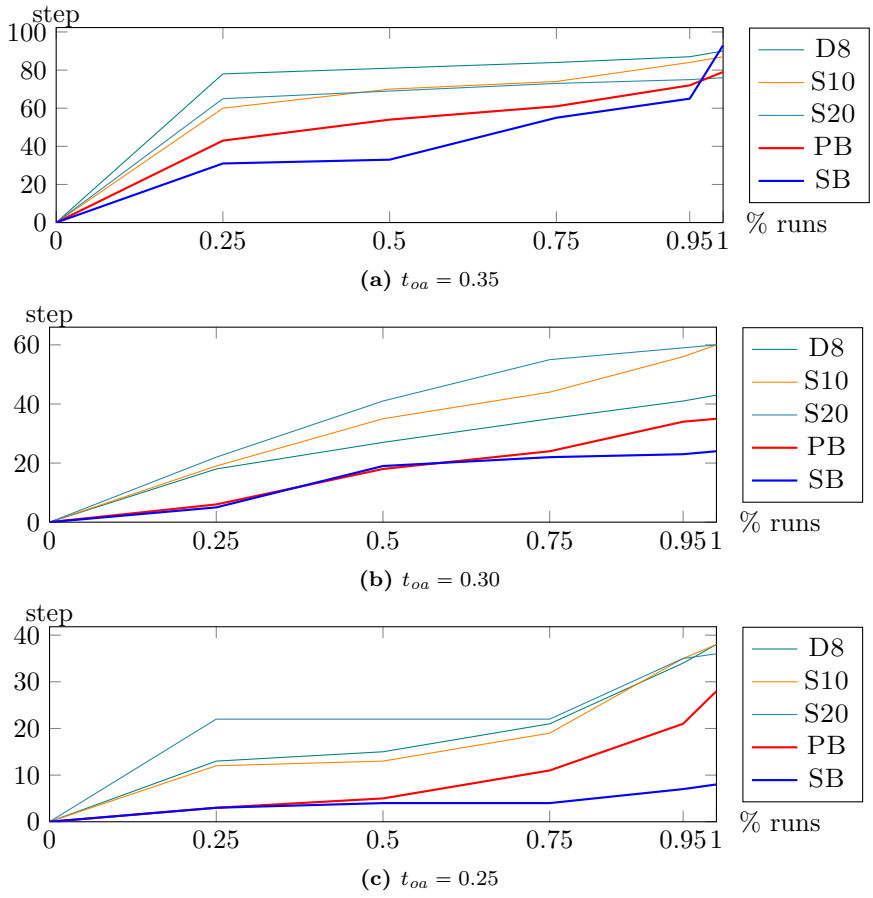


Figure 3.12: *The number of steps (y-axis) required by the tested heuristics to terminate the first (0.25), second (0.5), third (0.75) quartiles and the 95% (0.95 in the plot) of all the 84 executions (x-axis).*

in a 39% improvement over the best uninformed strategy (S10). For $t_{oa} = 0.25$, such improvement increases to 78% over S10. In other terms, in order to find one of the 17.8% optimal solutions, SB needs to explore, on average, only about 3% of the system configurations.

Interpretation. The results of our simulations *support* H_1: the informed heuristics converge quicker than uninformed heuristics, both in terms of number of steps and percentage of explored system configurations. This entails that probabilistic reasoning about monitored requirements seems to deliver an added value over the uninformed distance-based heuristics, in terms of convergence speed.

Quality (H_2)

Quality of the final solution. All the tested algorithms stop searching for new system configurations when a system configuration that meets the desired threshold is identified. As such, the average objectives achievement joint probability of the final solution is always above the threshold t_{oa}. All tested heuristics provide on average similar results in terms of average objectives achievement joint probability of the final solution; the *Final conf* column of Table 3.9 reports such value. For $t_{oa} = 0.35$, all algorithms provide an average value of the final solution that is close to the overall best possible solution: the quality of the final solution is, on average, 0.49, which is 0.14 higher than the threshold. The average quality of the identified solutions decreases with the lower values for t_{oa}, and we do not see differences between the algorithms. This is an expected behavior, for hill climbing algorithms employ local search techniques that stop as soon as an optimal solution is found.

Quality throughout the process. As described at the beginning of the section, we use different metrics for the analysis of the quality of the heuristics throughout the process of finding an optimal solution.

Fig. 3.13 illustrates the trend of the average objectives achievement during the optimization process, until all algorithms terminated all the 84 executions. A value in the plots for an algorithm A at step i represents the average value (w.r.t. all the 84 executions) of the objectives achievement joint probability oa of the configurations tried at step i.

If we consider metric $\overline{oa}_A$ (i.e., the average quality of the solutions tried by an heuristic A before stopping, reported in column $\overline{oa}_A$ of Table 3.9), the best heuristic is PB, which in all cases provides on average better solutions throughout the process (see also the red line in Fig. 3.13. When we consider, instead, the initial phases of the optimization process (the first 5-6 steps in the three sub-figures), we see that SB outperforms PB, selecting higher-quality solutions. This, as seen in Sec. 3.7.3 in the case of $t_{oa} = 0.25$, allows to find a solution above the threshold in very few steps. Due to this behavior, SB is always the fastest heuristics at reaching the desired threshold, outperforming the other strategies (and in particular the uninformed ones) also in terms of $\overline{oa}_{A,thres}$ (reported in column $\overline{oa}_{A,thres}$ of Table 3.9).

Fig. 3.14 reports the average $avg_{i=1}^{n} p_{Ai}$, where $p_{Ai} = avg_{i=1}^{84} oa_{Ai}$ is the average objectives achievement joint probability for the configurations tried by algorithm A at simulation step i, and n is defined as follows:

- Fig. 3.14a: the step when all algorithms terminate all executions: 92 for $t_{oa} = 0.35$, 59 for $t_{oa} = 0.30$, 37 for $t_{oa} = 0.25$.
- Fig. 3.14b: the step when the first algorithm ends all its executions, i.e., 75 for $t_{oa} = 0.35$, 23 for $t_{oa} = 0.30$, and 7 for $t_{oa} = 0.25$.
- Fig. 3.14c: the step when the first algorithm meets the threshold i.e., 34 for $t_{oa} = 0.35$, 19 for $t_{oa} = 0.30$, and 3 for $t_{oa} = 0.25$.

The bar charts in Fig. 3.14 confirm the superiority of the informed algorithms when considering the average objective satisfaction rate throughout the process. The improvements are visible in all conditions, and become manifest if we consider the step

when the first algorithm ends (metric $\overline{oa}_{A,first}$, illustrated in Fig. 3.14b and reported in column $\overline{oa}_{A,first}$ of Table 3.9): for $t_{oa} = 0.35$, the informed algorithms provide a gain of about 37-50% over the non-informed algorithms, and for $t_{oa} = 0.25$, the gain delivered by SB over the average of the uninformed algorithms is about 88%.

Concerning $\overline{oa}_{A,last}$ (Fig. 3.14a and column $\overline{oa}_{A,last}$ of Table 3.9), since SB terminates the majority of the solutions early in the optimization process, its value of $\overline{oa}_{A,last}$ (which includes in the average also the *oa* of the executions already terminated) is generally higher than other metrics. However, due to the overall high quality of the solutions explored by PB, in some cases its performance is slightly better than SB. In general, Fig. 3.13 highlights that PB exhibits a more stable behavior in terms of quality of explored system configuration, if compared to SB.

Interpretation. The results of our simulations *partially support* H_2: while the quality of the final solution is not affected greatly by the algorithm, the informed heuristics show a higher objectives achievement joint probability throughout the process of finding an optimal system configuration.

Stability (H_3)

As reported in Sec. 3.7.3, despite SB and PB are comparable when it comes to the average objectives achievement joint probability, PB exhibits a more consistent behavior than SB, which instead leads to more intense oscillations, due to the more heterogeneous definition of the neighborhood.

To better understand these differences and similarities, we focus on *stability metrics* in terms of the number of performed requirement revisions. This metric matters, for each revision may incur some costs (e.g., to deploy the necessary sensors for monitoring requirements compliance), which the system designer wants to minimize. Fig. 3.15 reports the trend of the average number of revision performed at each step of the optimization process. Note that the lines for an individual algorithm end when all the 84 executions of that algorithm terminate. The figure reports, at a given step i, the average number of revisions performed w.r.t. the executions that are still running at step i. Conversely, the executions that already terminated are not considered when computing the average. An approximation of the number of executions still running at a certain step i can be seen in Fig.3.12.

The total number of revisions for SB to find an optimal solution is lower than the other algorithms in all cases (see the *Total revs* column of Table 3.9). This is particularly evident with the lowest threshold $t_{oa} = 0.25$, since SB finds a solution in very few steps, leading to an average total number of revisions of 9.05.

Concerning the average number of revisions performed at each step (column *Revisions per step* in Table 3.9), note that all algorithms are comparable with thresholds $t_{oa} = 0.35$ and $t_{oa} = 0.3$, which require more steps than $t_{oa} = 0.25$. However, in the initial phases of the optimization process (first steps), SB performs an higher number of revisions per step, compared to the other algorithms. This explains why, with $t_{oa} = 0.25$, it reaches an optimal solution faster than the others.

Interpretation. The results of our simulations *support* H_3: the total number of revisions that informed heuristics make is lower than the number for uninformed

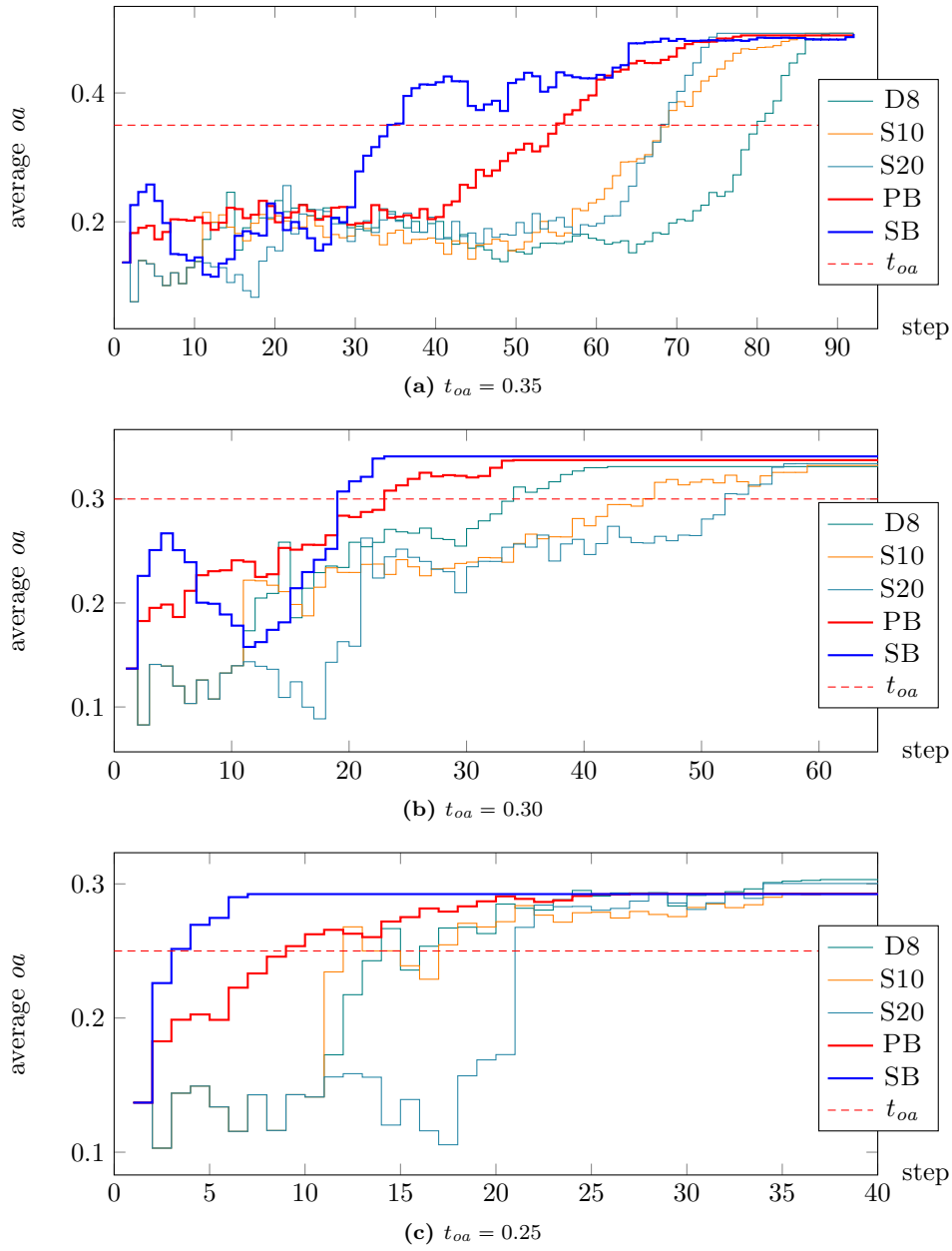
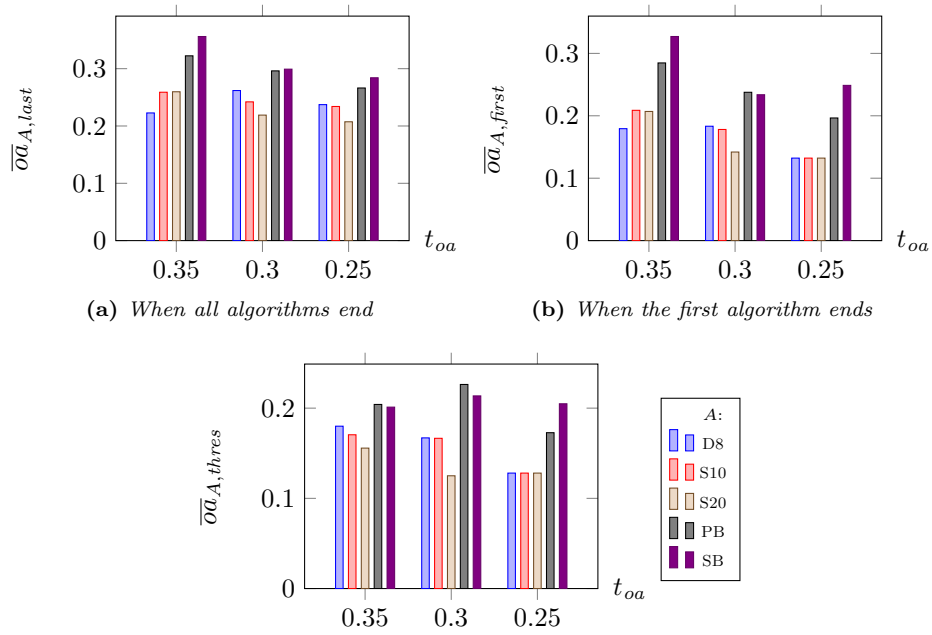


Figure 3.13: *The trend of the average objectives achievement joint probability (y-axis) of the solutions selected during the optimization process. The x-axis indicates the steps made by the hill climbing algorithm.*

heuristics. However, to do so, the informed algorithms make—in some cases—more revisions per step than the uninformed heuristics.



(a) *When all algorithms end*

(b) *When the first algorithm ends*

(c) *When the first algorithm meets the threshold*

Figure 3.14: *Average objectives achievement joint probability (y-axis) throughout the process of finding an optimal solution, for all the 3 tested thresholds (x-axis).*

3.7.4 Noise tolerance for the informed algorithms (H_4)

The performance of the informed heuristics depends on the quality of the data analysed. So far, we assumed our system can monitor the satisfaction of the requirements perfectly. We now relax this assumption to analyze how the algorithms perform in the presence of noisy data about requirements satisfaction.

We compare the performance when a certain percentage p of the information acquired from the *Monitoring* component is incorrect. In particular we analyze results with p =5%, 10%, and 20%. To do so, we modified our dataset by uniformly altering p% of the data concerning active requirements satisfaction. Specifically, we changed, with probability p, every value *obeyed* and *violated* in Table 3.3, respectively into *violated* and *obeyed*.

Fig. 3.16 reports the results for the variation of the percentage of explored system configurations by the two informed heuristics PB and SB when introducing noise. Even with 20% of noise, with $t_{oa} = 0.30$ and $t_{oa} = 0.25$, the algorithms presents almost no difference in the system configurations selected. In case of $t_{oa} = 0.35$, when more system configurations need to be explored, the maximum detected variation is of about the 1% in terms of system configurations explored, when perturbing 20% of the input data.

Fig. 3.17 shows the impact of noise on the average objectives achievement joint

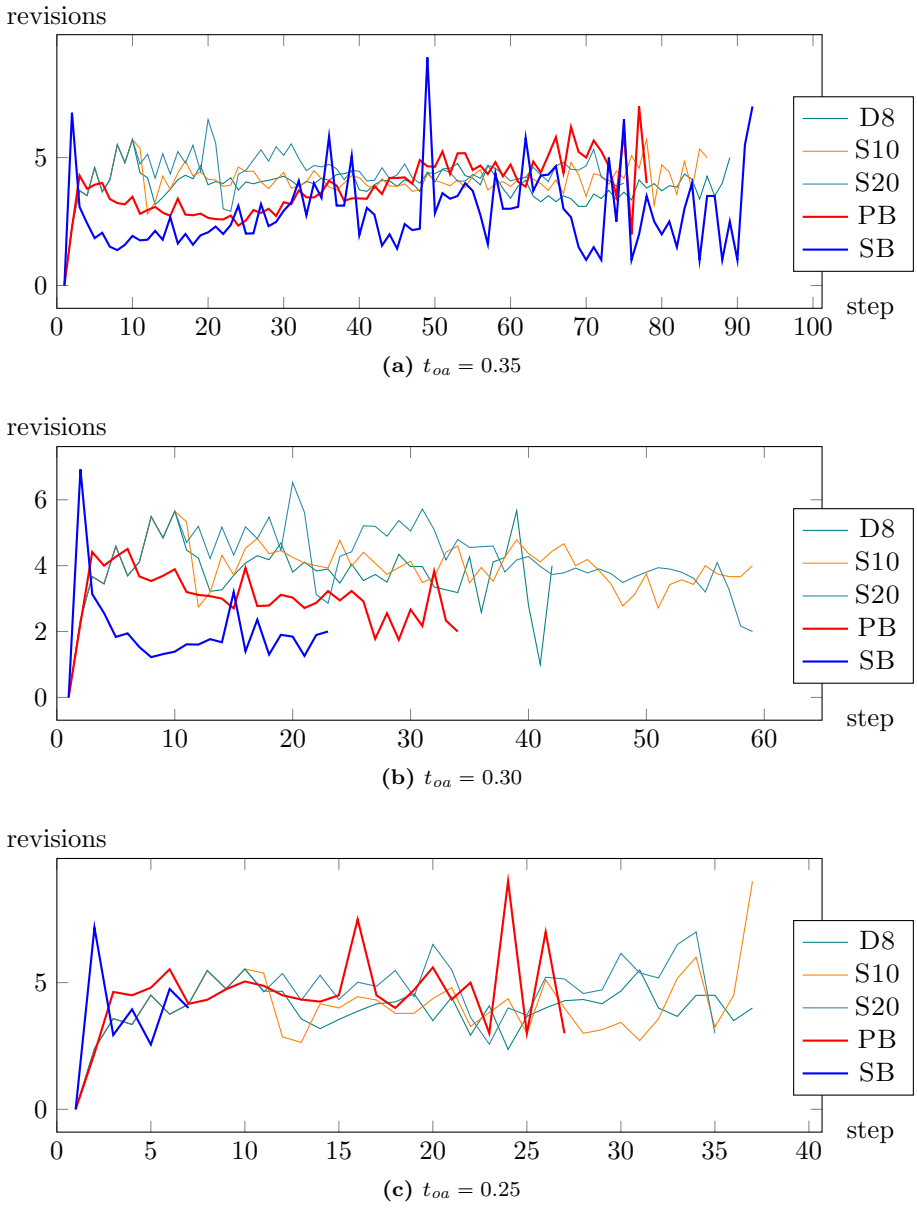


Figure 3.15: *Average number of revisions (y-axis) performed at each step (x-axis) of the optimization process.*

probability over the steps of the optimization process in case of $t_{oa} = 0.35$, the only threshold level at which the introduced noise had some noticeable impact. The figure shows no impact during the early phases of the optimization process, while the effect is visible after several steps of optimization, due to the presence of an increasing

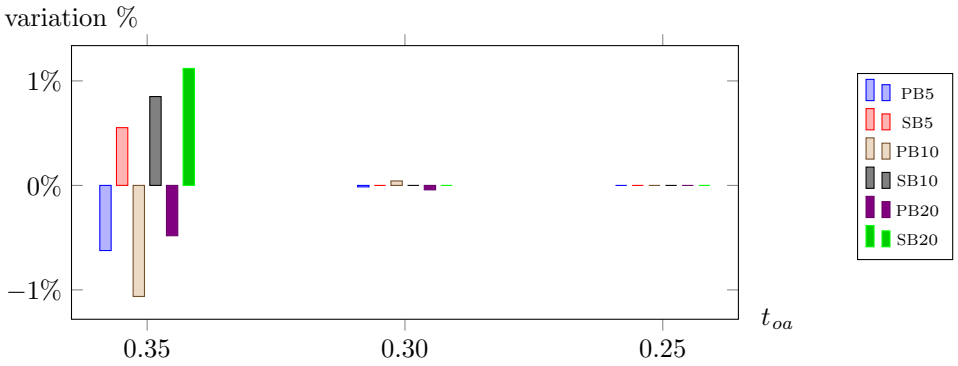


Figure 3.16: *The variation of the percentage of system configurations explored (y-axis) when introducing 5%, 10% and 20% of noise in the input data (respectively identified with* PB*5 and* SB*5,* PB*10 and* SB*10 and* PB*20 and* SB*20), for each of the 3 tested thresholds (x-axis).*

quantity of noisy data. The effects, however, are within the 2% range. The line chart also helps understand why the algorithms are not impacted with lower thresholds: the effect of noise occurs after multiple steps, while our algorithms return before such effects are visible.

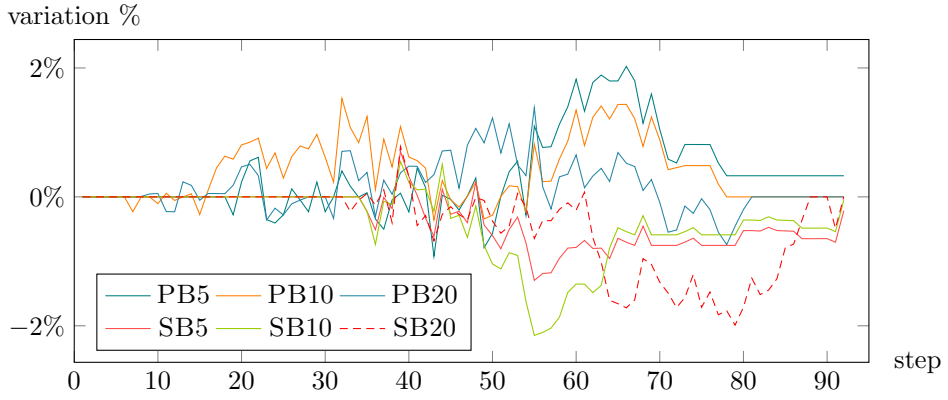


Figure 3.17: *The trend of the variation of the average objectives achievement joint probability (y-axis) over the steps of the optimization process (x-axis) in case of* $t_{oa} = 0.35$ *when noise is introduced.*

Interpretation. The results of our simulations *support* H_4: the informed heuristics seem to have high tolerance to degrees of noise up to 20%. However, it must be noted that data was perturbated in a uniform manner, and this may have positively affected the ability to tolerate noise.

3.7.5 Threats to Validity

The implementation of the prototype of the control loop described in Sec. 3.6.2 could be incorrect, which would render the results invalid. We reduced the potential impact of this threat by performing an extensive testing of the implementation and by applying it to different problems.

The chosen topology of the *Requirement Bayesian Network*, reflecting the structure of a requirements model, may influence the conclusions drawn via probabilistic inference. The choice of such topology is a threat to construct validity. For example, we do not capture causal relationships between sibling requirements or between objectives, which may help better explain when and why the requirements and the objectives are achieved. Different mappings between a requirements model may be tried to overcome this limitation.

The interpretation of the results is subject to the size and type of the set of system configurations tested. To mitigate this threat, we paid attention to our interpretation and the wording of the implications, and we deliberately omitted tests for statistical significance due to the use of a single case.

The notion of degree of validity is based on the assumption that the collected positive and negative evidence have the same statistical significance. The choice of such method to evaluate the assumptions affects construct validity. Additional probabilistic learning techniques should be explored and tested.

Finally, our conclusions have only limited generalizability. It is possible that the proposed algorithms behaves differently on different problems. This threat to conclusion validity is partly mitigated by our previous work [117], where we applied the same approach to a different problem, obtaining analogous results.

3.8 Related Work

The intrinsic dynamism of modern software systems leads to high run-time uncertainty [190, 294], which makes adaptation a necessity. Researchers argue for the necessity of evaluating the assumptions made during the design of a system in order to support software evolution. In their seminal works [190, 191], Lehman *et al.*, identify invalid assumptions as one the main causes for software evolution. They highlight how the—implicit or explicit—presence of assumptions in (E-type) software is inevitable and follows from the fact that real-world software and the environment in which it operates have a potentially unbounded number of properties.

Several approaches for the evaluation of assumptions have been proposed over the years. Boness *et al.* [55, 56] explicitly represent assumptions when defining requirements within a goal oriented framework. They use such concept to help the system designers to assess, during requirement analysis, the confidence in (and the risk due to) the set of elicited requirements. In order to do so they integrate expert knowledge, argumentation techniques and propagation of confidence values through the goal graph. In our work, we consider assumptions that are implicit in the structure of a requirements model (rather than explicitly represented as in [56]) and we calculate (and make use of) their degree of validity at run-time by means of probabilistic reasoning on a Bayesian Network trained with data obtained during the execution.

The availability of a requirements model during execution [40, 48] is crucial to build a framework that supports the run-time evolution of the system requirements. Several frameworks exist that use such models to support the monitoring and diagnosis of requirements [140, 236, 290]. Such approaches are powerful and allow the identification of deviations from the requirements. However, they do not challenge the validity of the requirements (models) themselves.

Ali *et al.* [15] illustrate the advantages of monitoring requirements at run-time to detect when design-time assumptions concerning requirements satisfaction become invalid. They also discuss the importance of keeping track of the relationship between context and requirements at run-time [16]. Paucar *et al.* [226] propose techniques to reassess the assumptions about the priority of non-functional requirements.

Models at run-time are often used for guiding the adaptation of the system. Souza *et al.* [260] define awareness requirements as meta-requirements to drive adaptations. Non-functional requirements (NFRs) have been used to trigger and guide self-adaptation; for example, contributions to objectives can help identify those system configurations that maximize NFR satisfaction [100, 241].

These approaches constitute our baseline: our framework uses requirements models at run-time [40, 48], can rely on existing monitoring frameworks [236, 290], and implements the idea of reconsidering design assumptions at run-time [15]. The distinguishing features of our approach are the focus on socio-technical systems, the use of Bayesian learning, and the employment of an hill climbing approach to identify an optimal system configuration.

In order to support the automated requirements evolution, Whittle *et al.* [294] propose the notion of requirements revision. They present a requirements language for self-adaptive systems (RELAX) that allows to specify relaxed versions of a requirement during the elicitation phase. Existing requirements revision approaches mainly focus on re-assessing the weights of non-functional requirements [17, 39]. Knauss *et al.* [177] discuss the mining of optimal contexts for previously defined contextual requirements, and propose a revision of the contextual condition of applicability of the requirements.

The normative multi-agent systems (NMAS) literature offers techniques for the dynamic update of norms that regulate a multi-agent systems. Aucher *et al.* [25] introduce a dynamic context logic that describes the operations of contraction and expansion of theories by introducing or removing new rules. Governatori *et al.* [156] investigate the legal consequences of applying theory revision to reason about legal abrogations and annulments. Alechina *et al.* [10] show how to formally obtain an approximated version of a norm to cope with imperfect monitors for the original norm. Since norms are an important type of requirements for STSs [81, 256], NMAS research is a rich cross-fertilization tool for the RE discipline.

In previous work [118], we proposed Bayesian Networks as a tool to learn, from run-time data, the correlation between the satisfaction of requirements and the achievement of the system's objectives in different operating contexts. In [118], we show that such information can be used to validate some assumptions made in a goal model. In this chapter, we embedded our requirements assumptions validation technique within a holistic framework for the evolution of STSs; in particular, the validity of the assumption is used by our heuristic algorithms that perform run-time requirements

revision.

Cailliau *et al.* [71] present a technique for the quantitative assessment of requirements-related risks. Their framework uses KAOS goal models extended with a probabilistic layer to evaluate the consequences of explicit obstacles on the satisfaction of goals. The concept of obstacles is similar to the concept of harmful requirements presented in this chapter; however, we do not consider requirements that are known to be harmful *a priori*. We focus, instead, on techniques for discovering at run-time whether some requirements are useful or not and in which contexts.

Our requirements revision types (relaxation, strengthening, disabling, etc.) are similar to the strategies presented by Lamsweerde *et al.* [276] to resolve goal conflicts. In their work, such strategies are applied in the early stages of requirements elicitation (at design time) and they rely on the available domain knowledge. Our framework focuses on the run-time analysis of the requirements and of the assumptions made at design time. A deeper study of the relationship of our work with run-time conflict resolution techniques is left for future work.

3.9 Discussion and Future Work

We introduced a novel framework for guiding the evolution of Socio-Technical Systems. Our approach uses requirements models to represent system's objectives, requirements, and their relationships. The framework supports both the *manual evolution* of the STS, by revealing the validity of the assumptions in a requirements model, and the *automated adaptation*, by revising the requirements in order to quickly identify an optimal system configuration.

This work employs two techniques from artificial intelligence: (i) *Bayesian Networks* as a tool to learn and reason about the relationship between contexts, requirements and objectives based on evidence from system execution; and (ii) *hill climbing algorithms* as a technique to explore the space of alternative configurations of the STS and identify optimal configurations that maximize the satisfaction of the objectives.

Our experiments with a smart traffic simulator show promising results for our automated requirements revision algorithms. Both the PB and SB heuristics that we propose outperform uninformed hill climbing heuristics. The requirement revisions are guided, in our algorithms, by the information retrieved from run-time execution data about the validity of the assumptions made in a requirements model. The results show that using that information allows to accelerate convergence to an optimal configuration by guiding the requirement revision process. Our heuristics provided, in certain cases (see Sec. 3.7.3), an efficiency gain of about 75% compared to uninformed heuristics, in terms of number of the explored configurations of requirements. The heuristic SB was able to terminate 95% of all its executions before all the baseline uninformed heuristics could reach their first quartile. In one experimental setting, SB was able to find, on average, an optimal configuration in less than 2 steps, exploring about 3% of possible configurations of requirements in order to find one of the 17.8% optimal ones.

The results revealed that our informed algorithms positively affect the quality of the attempted system configurations. When considering the average stakeholders' ob-

jective satisfaction rate throughout the optimization process, our informed heuristics provided an improvement, compared to uninformed ones, ranging from 37% to 88% (see Sec. 3.7.3 for more details).

Our analysis of the stability of the algorithms, in terms of number of revisions, showed also that our informed heuristic SB suits well those problems for which an optimal solution needs to be found quickly with a small total number of revisions along the process. Compared to the other tested algorithms, however, SB includes steps in which it performs a high absolute number of revisions (see, for instance, the initial peaks in Fig. 3.15). Should there be a limit on the maximum number of acceptable requirement revisions, other heuristics could be preferable. A possible reason why this factor may matter is that revising requirements may pose some challenges for humans to adapt to the new requirements (e.g., think of revising the speed limits of all streets at the same time).

Finally, our proposed algorithms exhibited a high tolerance to possible noise in the data used to train the Bayesian Network: a uniform perturbation of 20% of the input data by introducing erroneous information about requirements satisfaction lead only to a variation of about 1% in terms of number of system's configurations explored during the optimization process and impacted less than 2% on the average stakeholders' objectives satisfaction.

Limitations and Future work. A thorough evaluation of the scalability, usefulness and generality of our proposal is imperative. So far, we have focused on smart traffic simulations because this is an example of an STS that has numerous simulator frameworks. Our current Bayesian Network assumes a consistent behavior of the STS population over time. *Dynamic Bayesian Networks* [240] should be considered to support more dynamic STSs, in which we cannot make such assumption. Furthermore, the two revision algorithms that we introduced do not store any information concerning the effects of the requirement revisions applied. Refined revision algorithms shall be developed with a larger memory than just the current configuration; possible techniques include Q-Learning [239] and Dynamic Decision Networks [240]. Moreover, we plan to develop algorithms that can guide software evolution by providing additional information on the most critical and significant assumptions. To do so, we plan to employ other analysis techniques for Bayesian Networks, such as sensitivity analysis [274] or qualitative reasoning [292]. In Chapter 4, for example, we use sensitivity analysis to provide a quantitative estimation of the required strength of the sanctions needed to motivate the agents to comply with the norms. Visualization tools, which are missing in this chapter, are necessary to support human analysts in visualizing the validity of the assumptions in a requirements model, as discussed in [234], and to help them in deciding about the manual evolution of an STS. A starting point could be the visualization we developed in earlier work [118]. Finally, while this chapter focuses on the revision of requirements by exploring the space of alternatives within a model, in the next chapters, and in particular in Chapter 5, we will explore the possibility to synthesize new requirements that are not included in the given model.

4 Data-Driven Run-Time Revision of Sanctions

One way to achieve the system-level objectives of a multi-agent system without limiting the autonomy of the individual agents is to control their behavior by enforcing norms by means of sanctions. The dynamicity and unpredictability of the agents' interactions in uncertain environments, however, make it hard for designers to specify norms and sanctions that will guarantee the achievement of the system-level objectives in every operating context. In this chapter, we extend the supervision mechanism for the run-time revision of norms presented in the previous chapter, to support also the revision of the sanctions of the norms. This chapter aims at tackling some limitations in the research literature of run-time sanctions revision and concerning in particular with the lack of run-time approaches that do not rely on knowledge of the internals of the agents and that, in determining optimal sanctions, do not assume a priori that violations are detrimental for the system-level objectives. As in Chapter 3, we use a Bayesian Network to learn from system execution data the relationship between the obedience/violation of the norms and the achievement of the system-level objectives. In addition to run-time data about the behavior of agents, we leverage here also some knowledge about the preferences of rational agents. We devise heuristic strategies that combine the knowledge acquired at run-time with an estimation of the preferences of rational agents to automatically determine new sanctions that are expected to improve the achievement of the system's objectives. We evaluate our heuristics using a traffic simulator and we show that our mechanism is able to quickly identify optimal revisions of the initially enforced norms.

This chapter has been published in:

- Dell'Anna, Davide, Mehdi Dastani, and Fabiano Dalpiaz. "Runtime revision of sanctions in normative multi-agent systems". *Autonomous Agents and Multi-Agent Systems*, 34.2 (2020): 1-54.

This chapter provides an answer to research questions **RQ 2-4** from Chapter 1 in the context of revision of sanctions in MASs.

Acknowledgement. We thank Dr. S. Renooij for her advice and support on the issues in this chapter related to sensitivity analysis.

4.1 Introduction

Multi-Agent Systems (MASs) comprise autonomous agents that interact in a shared environment [297]. To achieve the system-level objectives of a MAS, the behavior of the autonomous agents should be controlled and coordinated [68]. For example, a smart traffic system is a MAS that includes autonomous agents like cars, traffic lights, etc. The objectives of the system include avoiding the occurrence of traffic jams as well as minimizing the number of accidents.

One way to control the behavior of the agents in a MAS without limiting their autonomy is norm enforcement [11, 265]. Norm enforcement via sanctions is traditionally contrasted with norm regimentation; the latter alternative prevents the agents from reaching certain states of affairs. For example, in a smart traffic system, a regimentation strategy is to close a road to prevent cars from entering that road, while a sanctioning strategy is to impose sanctions on cars that drive through the road.

Due to the dynamicity and unpredictability of the behaviors of interacting agents in uncertain environments, it is difficult for the designers who engineer a MAS to specify norms that, when enforced, will guarantee the achievement of system-level objectives in every operating context. To cope with this issue, the enforced norms need to be revised at run-time. Existing research has investigated the offline revision of the enforced norms[10], proposed logics that support norm change [25, 179, 180], and examined the legal effects of norm change [156].

In [117], we proposed a framework for engineering normative MASs that, using observed data from MAS execution, revises the norms in the MAS at run-time to maximize the achievement of the system's objectives. In that work, we made the simplistic assumption that norms are regimented and we introduced algorithms for switching among alternative predefined norms. In [119], we extended the framework to support the revision of norm enforced via *sanctioning*. In addition to observed data from MAS execution, we used an estimation of the preferences of the agents to guide the run-time norm revision. However, we considered MASs where only one norm at a time was enforced.

In this chapter, we significantly extend our previous work by supporting MASs where multiple norms are enforced. We formalize different types of rational agents that behave according to their preferences and we disccus their properties. We use Bayesian Networks to learn the norm effectiveness from data observed from MAS execution and to inform the run-time norm revision mechanism that revises the sanctions of multiple norms.

The contributions of this chapter are as follows:

- We provide a formal definition of different types of rational preferences of agents, specified in terms of desired states of affairs and the maximum payment that the agent is willing to make to achieve such states of affairs. We prove that such preferences satisfy the basic rationality requirements [202].

- We build on and extend the general architecture proposed in [117, 119], and study in detail the relationships between estimated agents' preferences, sanctions, and system-level objectives. We use a framework where the normative MAS is flanked by a norm monitoring and enforcement component, and we

introduce a norm revision component that uses observed data from MAS execution and an estimation of agents' preferences to modify norm sanctions at run-time.

- We propose six heuristic strategies for the revision of multiple norms that leverage probabilistic information learned from observed data from MAS execution and an estimation of the preferences of agents.

- We report on an evaluation through a traffic simulator that shows the effectiveness and efficiency of our revision strategies in identifying optimal sanctions for multiple norms.

Organization. Sec. 4.2 reports on related work. Sec. 4.3 presents our framework to characterize norms and agents' preferences. Sec. 4.4 explains the overall approach for the supervision of normative MAS based on probabilistic reasoning over norm effectiveness and agents' preferences. Sec. 4.5 introduces six strategies for revising norms by combining agents' preferences with the achievement of the system-level objectives. Sec. 4.6 evaluates our work through simulation experiments. Sec. 4.7 discusses the results and the assumptions, limitations and future directions of our work. Sec. 4.8 presents our conclusions.

4.2 Related Work

In the MAS literature, norms have been proposed as a way to regulate the behavior of the agents in order to achieve system-level properties without limiting the autonomy of the agents [11, 265, 284].

Many approaches focus on the design-time construction of robust normative MASs. Several techniques enable proving the correctness of normative systems through the model checking of formulas that describe liveness or safety properties [9, 109, 178]. These works are useful for the initial design of a MAS, but they cannot cope with the run-time unpredictability of the system that stems from the autonomy and heterogeneity of the agents.

In order to successfully supervise and regulate dynamic MASs, researchers have studied the revision of norms. Some frameworks formalize norm dynamics thereby allowing the assessment of the impact of norms on the specification of a MAS, i.e., whether the designed MAS will be norm compliant. Aucher *et al.* [25] introduce a dynamic context logic to describe the operations of contraction and expansion of theories that occur when removing or adding new norms. Governatori *et al.* [156] investigate how the application of theory revision leads to legal abrogations and annulments. Knobbout *et al.* [180] propose a dynamic logic to characterize the dynamics of state-based and action-based norms. Both in Knobbout's work [179, 180] and in Alechina *et al.*'s approach [9], norm change is restricted to norm addition. This family of approaches focus on the impact of revising a norm on an existing normative system. In this chapter, instead we study the relationship at run-time between the enforced norms and the achievement of system-level objectives, and suggest mechanisms to determine *how* to revise the (sanctions of the) current norms.

Jiang *et al.* [167] discuss the contextualization of norms. They explicitly represent the context of application of a norm and they use such context to organize norms during the design of a MAS. In our work, we also enforce different norms in different contexts. Unlike them, however, we determine the most appropriate context for different norm sets at run-time and based on observed data from MAS execution.

Miralles *et al.* [208] present a framework for the adaptation of MAS regulations at run-time. Their approach is complementary to ours. They represent conditional norms via norm patterns and describe an adaptation mechanism based on case-based reasoning. Adaptation is performed at run-time individually by a number of assistant agents and then, via a voting mechanism, a final adaptation is approved. The decision on how to adapt norms is taken based on similar previously seen cases. In their work, however, they do not consider sanctions. In our work, we focus on the revision of sanctions, we perform norm revision through a centralized component, and we make use of an estimation of agents' preferences to guide norm revision.

Cardoso *et al.* [72] present a framework for the run-time adaptation of sanctions associated with obligations. In their work, they assume that norm violations are bad for the system-level objectives. In our work, we relax such assumption, as agents ability to violate norms can be useful [75]. We evaluate the effectiveness of a norm at run-time based on observed data from MAS execution. Furthermore, they assume that the strength of a sanction should be directly proportional to its application frequency, and they constantly try to lower sanctions in order to give agents maximum autonomy. In our work, we base the revision of norms on an estimation of the preferences of the agents, and we determine the appropriate value of their sanctions based on the relationship between obedience of norms and achievement of system-level objectives determined at run-time.

In MASs, agents' preferences have been mainly used as a way to choose at run-time between different plans or actions to execute [103, 168, 223, 286]. Preferences are usually interpreted as constraints that, if satisfied by a certain plan (or action), increase the desirability of executing such plan (or action). Formal languages have been proposed and used for expressing preferences (e.g., $\mathcal{LPP}$ [29, 47] or *LTL* [68]). In this chapter, we focus on strategies for sanctions' revision. For this reason, we make use of a high-level representation of preferences, without restricting ourselves to, but supporting, any specific language. In particular, we consider preferences that satisfy the basic rationality requirements [202] and order different alternative states of affairs that agents may desire to achieve. Our agents are rational and norm-aware [279], in the sense that they always try to aim at the most preferred state of affairs for which they have enough budget, taking also into account the possible sanctions they would incur when violating some of the enforced norms. Furthermore, our agents are autonomous, in the sense that they are able to make decisions without the intervention of human users but in line with their preferences [27, 107]. As we aim to investigate the process of norm revision, we assume that we have an accurate estimation of the agents' preferences. In future work, we can relax this assumption and investigate norm revision based on inaccurate estimations of the agents' preferences.

Chopra *et al.* [80] study how agents' preferences—expressed in terms of goals—interact with norms—represented as commitments. In particular, they propose a framework for the agents to adapt their behavior. We take an orthogonal approach,

for we study how to change the norms without altering the agent construction. In particular, we study how to alter the sanctions used to enforce the norms on the agents, so to guarantee at run-time the system-level objectives. Our proposed mechanisms, therefore, relate also to the idea of adjustable autonomy [215]. The proposed run-time mechanism of revision of the sanctions of the norms can be seen as an automated mechanism to adjust the decisions' options of the agents (thus their degree of autonomy) so to maximize the objectives of the system and its operators.

Cranefield *et al.* [95] present a Bayesian approach to norm identification. They show that agents can internalize norms that exist in an environment, by learning from the behavior that complies with or violates certain norms. This work is a valuable addition to ours, for it shows that it is possible for agents to learn norms even when they are not explicitly communicated to them.

Tumer *et al.* [269] use multi-agent reinforcement learning in a smart traffic simulation to determine the behavior of the car agents that maximizes the utility of the city designer and of the individual agents. Their interesting work focuses on regimentation; instead, we focus on enforcement that does not violate agents' autonomy.

4.3 Normative Multi-Agent Systems

This section presents a generic framework for specifying normative multi-agent systems in which the agents behave in line with their preferences while norms are enforced on them via sanctions. This framework allows us to analyze the interplay between norms and agents' preferences in normative multi-agent systems.

4.3.1 Illustrative example

Consider the two-lanes ring road depicted in Fig. 4.1. In a ring road, a population of vehicles moves continuously in a circle. Every vehicle is autonomous and acts according to its own preferences. For example, vehicles have preferences about, among other things, their speed, based on which they determine their willingness to risk sanctions for violating traffic norms. Such preferences and their corresponding willingness to risk sanctions allow the vehicles to autonomously decide when and how to accelerate or decelerate or to change lane. If a fast vehicle is using the outer line and a slower vehicle blocks its way, the fast vehicle may move to the inner line to overtake the slow vehicle. Since all vehicles share the same environment, their local decisions have an effect on the (emergent) system-level behavior of the vehicles driving on the ring road [262]. For example, based on contextual factors such as the density of vehicles on the ring road, the vehicles' behavior may provoke traffic jams and the average speed may vary, as well as the average time to complete a loop of the ring road. The ring road is a simple example of a MAS. Although far from realistic traffic situations, the ring road illustrates the fundamental phenomena of emergent system-level properties, caused by the local decisions of individual agents, and the importance of mechanisms to control and steer such system-level behaviors.

We assume that the main stakeholder of the ring road (the city council) has two system-level objectives: *to minimize the average time to complete a loop of the ring road* and *to minimize the number of halted cars*. Despite interdependence, the

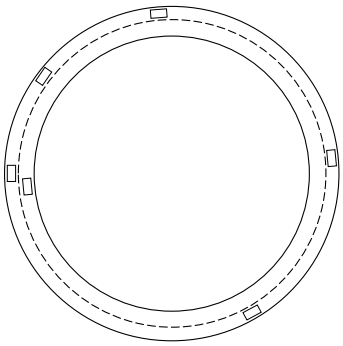


Figure 4.1: *Two lanes ring road. Rectangles are vehicles, moving in counter clockwise direction.*

stakeholder desires to evaluate the two objectives independently due to their distinct nature. We consider two contextual variables that may influence the achievement of the system-level objectives, together with the vehicles' behavior: the *density of vehicles* and the *presence of an obstacle* on the ring road. The higher is the density of the vehicles on the ring road, the higher is the risk of traffic waves and slowdowns. The presence of an obstacle may force vehicles to halt and wait for an adequate moment to take over the obstacle. If the density of vehicles on the ring road is high enough, this may also cause queues after the obstacle. To achieve the objectives, the behavior of the agents is regulated by enforcing norms concerning (i) the speed limit, such as the norm *every vehicle on the ring road shall not exceed a speed of 50km/h, otherwise it will receive a sanction of 100€*, and (ii) the minimum safety distance between cars, such as the norm *every vehicle on the ring road shall keep a minimum distance of 2m, otherwise it will receive a sanction of 20€.* Regulating speed and safety distance of the cars on the ring road is expected to help achieving the system-level objectives in the traffic contexts represented by the contextual variables. A car that keeps a sufficient safety distance from the car ahead, is less likely affected by sudden deceleration of the car ahead. More space between cars may also favour surpasses of slow cars when necessary. An opportune safety distance, together with opportune cars speed, may reduce jams in the presence of obstacles or the effect of traffic waves.

The ring road described above is a normative MAS. Vehicles are autonomous agents, each acting according to their own preferences. Each agent belongs to an agent type that can be characterized by the agent's preferences. For instance a *cautious* agent is a type of agent that prefers to go slow rather than fast on the ring road and prefers to maintain the appropriate safety distance. A *brave* agent is a type of agent that prefers to go fast rather than slow, and to approach cars closer than the minimum safety distance, even if it has to pay some money to do so.

4.3.2 Norms

The focus of this chapter is the run-time revision of the sanctions of the norms enforced in the MAS. In order to focus on this aspect, we propose a simple but extensible

language for norms. Consider a set of propositional atoms $L = \{p_1, \ldots, p_k\}$, each representing a fact that can hold or not in a system state[1] (e.g., propositional atom sp_{100} indicates that *the speed of a vehicle on the ring road is* $\leqslant 100$ *km/h*).

Let $AL = (L_1, ..., L_n)$ be an ordered list of n disjoint subsets of L, s.t. L_i contains atoms related to an aspect i of the system[2] (e.g., $L_i = \{sp_{100}, sp_{50}\}$, in the ring road scenario, contains atoms related to the *speed of the cars*).

We consider a norm as a pair $N = (p, s)$, where $p \in L$ and $s \in \mathbb{N}$, indicating that p should hold in the current system state for all agents, otherwise sanction s will incur. For instance, a norm $N = (sp_{50}, 100)$ indicates that *every vehicle on the ring road shall not exceed a speed of 50km/h, otherwise it will receive a sanction of 100€.*

In the following we consider an ordered set of norms $\mathcal{N} = \langle N_1, N_2, \ldots, N_n \rangle$ and assume that (i) norms are non-conflicting, i.e., obeying a norm N_i does not prevent an agent from obeying or violating any other norm in $\mathcal{N}$; and (ii) each norm regulates a different aspect of the system, so that the i-th norm $N_i = (p, s)$ in $\mathcal{N}$ is a pair where $p \in L_i$ (with L_i i-th set in AL) and $s \in \mathbb{N}$. For instance, if $AL = (L_1, L_2)$, $L_1 = \{sp_{50}, sp_{100}\}$ and $L_2 = \{dist_1, dist_2\}$, then $N_1 = (sp_{50}, 100)$ is a norm concerning the speed limit and $N_2 = (dist_2, 100)$ is a norm concerning the minimum safety distance.

Note that, despite these assumptions, norms can still influence each others by means of the behavior that they cause on the agents. For instance, if the density of vehicles on the ring road is high, in order to obey a norm concerning the minimum safety distance from the car ahead, an agent may need to decrease its speed, therefore obeying also a norm concerning the maximum speed limit. We distinguish, however, such influence from the concept of conflict, in the sense that the norm concerning the minimum safety distance does not prevent, *a priori*, an agent to either obey or violate the norm concerning the maximum speed limit, and vice-versa.

4.3.3 Rational Agents and Their Preferences

In MASs, agents are often assumed to be autonomous and possibly heterogeneous. Moreover, it is common to assume that the internal states of the agents such as their beliefs, preferences, and decision making mechanisms are unknown or partly known to other agents or to the institutions that regulate their behavior. In line with the theory of economic rationality [202], in this chapter we consider rational agents that behave according to their rational preferences, which determine an ordering between different alternative states of affairs (simply *alternatives* in the following). A rational agent aims to achieve its most preferred states of affairs: when a rational agent believes it is possible to achieve a certain state of affairs s, the agent will never aim to achieve states of affairs that are less preferred than s. For example, a *cautious* agent that prefers to go slow on the ring road and maintain appropriate safety distance, may be less prone to surpass other cars or to change lane, and may exhibit more moderate

[1] A system state is assumed to consist of the state of individual cars (e.g., speed and position of the cars) as well as the state of the environment (e.g., density of vehicles in the ring road).

[2] We use the term *aspect* to indicate any particular characteristics of the behavior of the agents, such as the speed of the cars, that is both monitorable by an organization that enforces norms in the MAS, and over which agents have control.

acceleration or deceleration than less cautious agents. The behavior of such a cautious agent, however, can vary significantly, based on contextual conditions. For example, a sudden break from the car ahead may force also the cautious agent to brusquely decelerate.

In this chapter, we assume we have an estimation of the preferences of the agents concerning the n different aspects of the system that we aim to regulate by a norm, as per Sec. 4.3.2. In the rest of the chapter, when we refer to the preferences of the agents, we refer therefore to such an estimation of their preferences. We do not assume access to the agents' internals such as their beliefs or their preferences regarding other aspects of the system (e.g., information about fuel reserve or the preference on road types). Having an estimation of the preferences of the agents should not be seen as a violation of the autonomy of agents or access to their internals. Having some knowledge of agents' preferences is realistic in most MAS settings. For example, in some cooperative settings, agents may be requested to declare their true preferences prior entering the system and agents can autonomously decide whether to join or not, while in other settings the preference of agents can be learned from their behaviors [38]. Note that we do not focus on the process of preference elicitation, which is essential for deriving and formulating agents' preferences, but beyond the scope of this chapter. Several techniques for the elicitation of preferences have been proposed in the literature, including both automated methods and methods that directly involve the end-user (see for example [61, 77, 248]). Here, we rely on such techniques and we just assume that some relevant part of agents' preferences is already given or estimated.

We represent the alternatives over which the agents have preferences as lists of pairs such as $(\langle p_1, b_1\rangle, \ldots, \langle p_n, b_n\rangle)$, indicating that for a state of affairs where $p_1, ...,$ and p_n hold, the agent is willing to spend, if necessary, a budget b_1 to achieve p_1, a budget b_2 to achieve p_2, etc. We focus on finite preferences, therefore we constrain the budgets expressed in the alternatives to be member of a budget set $\mathcal{B} \subset \mathbb{N}$.

We denote by $Pref(a) = (A, \succeq)$ the preference of an agent $a \in Ag$, where $Ag = \{a_1, \ldots, a_n\}$ is a set of agents, A is a set of alternatives defined as per Def. 9, and $\succeq$ is a partial order on A. We write $x \succeq y$ to denote the fact that the agent either prefers alternative x to alternative y or is indifferent between x and y.

Definition 9 (Preference Alternatives)**.** *Let $AL = (L_1, \ldots, L_n)$ be a list as per Sec. 4.3.2. Given a set of budget lists $BL \subseteq \mathcal{B}^n$ (with $\mathcal{B}^n$ the n-ary Cartesian power of $\mathcal{B}$), the set of alternatives A is the set $\{ (\langle p_1, b_1\rangle, \ldots, \langle p_n, b_n\rangle) \mid p_i \in L_i \ \&\ (b_1, \ldots, b_n) \in BL \}$.*

Notation. Before continuing, we provide here a summary of the notation that we will use in the rest of the chapter in the context of preferences. Given a preference $Pref(a) = (A, \succeq)$, an alternative $x = (\langle p_1, b_1\rangle, \ldots, \langle p_n, b_n\rangle) \in A$, and a set of budget lists $BL \subseteq \mathcal{B}^n$, we call:

- $prop(x) = (p_1, ..., p_n)$, the list of propositional atoms in x
- $bud(x) = (b_1, ..., b_n) \in BL$, the list of budgets associated to each propositional atom in x.

- $req\_bud(x) = \sum_{b \in bud(x)} b$ the budget required by alternative x (*required budget*, in the following), i.e., the sum of all budgets in x.
- $x[B']$ a new alternative $x' = (\langle p_1, b'_1 \rangle, ..., \langle p_n, b'_n \rangle)$ with same propositional atoms as x, but using budgets $B' = (b'_1, \ldots, b'_n) \in BL$ instead of budgets $B = (b_1, \ldots, b_n)$.

Furthermore, in the rest of the chapter, unless specified otherwise, when we provide an example concerning preferences or norms, we make use of L defined as the set $\{sp_{50}, sp_{100}, dist_1, dist_2\}$ with $AL = (L_1, L_2)$ and $L_1 = \{sp_{50}, sp_{100}\}$ and $L_2 = \{dist_1, dist_2\}$ so that $\mathcal{N} = \langle N_1, N_2 \rangle$ with L_1 related to N_1 (norm concerning speed limit) and L_2 related to N_2 (norm concerning safety distance), and we use n to indicate the number of norms in $\mathcal{N}$.

In the following we define the types of preferences that we consider in this chapter. We first define two basic types of preferences. Then, after providing some examples of such preferences, we define more complex preferences that combine the two basic types.

Basic Preferences

We define here two types of basic preferences. The first kind of preference orders the alternatives based on their budgets, while the second type orders the alternatives based on the propositional atoms (i.e., states).

Definition 10 (Basic Preference)**.** *Given a list $AL = (L_1, ..., L_n)$ and a set $BL \subseteq \mathcal{B}^n$, an agent is said to have a basic preference $(A, \succeq)$ when for all alternatives x and y in A, the partial order $\succeq$ satisfies one of the following two clauses:*

a. $x \succeq y$ *iff*
$req\_bud(x) \leqslant req\_bud(y)$ &
$\forall v, w \in A, \forall B, B' \in BL : v[B] \succ w[B] \Rightarrow v[B'] \succ w[B']$

b. $x \succeq y$ *iff*
if $prop(x) = prop(y)$ *then* $req\_bud(x) \leqslant req\_bud(y)$
else $\forall B, B' \in BL : x[B] \succeq y[B']$

In the rest of the chapter, we write $x \sim y$ when $x \succeq y$ and $y \succeq x$. We write $x \succ y$ when $x \succeq y$ but not $y \succeq x$.

If an agent's preference adheres to Def. 10a, then the required budget determines the order of the alternatives. In particular, Def. 10a determines a preference where alternatives that require a lower budget are preferred to alternatives that require higher budget (first condition of Def. 10a) and the relative order between two alternatives with different propositional atoms is the same for all possible budgets (second condition of Def. 10a). Note that in a basic preference that adheres to Def. 10a, two alternatives x and y such that $req\_bud(x) > req\_bud(y)$ cannot be equally preferred.

In fact, if $x \sim y$ we have that $req\_bud(x) \leqslant req\_bud(y)$ and $req\_bud(y) \leqslant req\_bud(x)$. As a consequence, all alternatives with required budget 0 are strictly preferred to all the other alternatives, and all alternatives with same required budget are equally preferred.

If an agent's preference adheres to Def. 10b, then the propositional atoms determine the order of the pairs. If a set of propositional atoms is preferred to another, then it is preferred regardless of the required budget. In a preference that adheres to Def. 10b though, the alternatives with required budget 0 are strictly preferred to all the other alternatives with same propositional atoms.

We would like to emphasize that the basic preferences as we defined here are different than lexicographic ordering [145]. An agent's preference, as per Def. 10, satisfies, instead, the basic rationality requirements [202], as per Prop. 1.

Proposition 1. *A basic preference* $Pref(a) = (A, \succeq)$ *for an agent* $a \in Ag$ *is*

- *transitive:* $\forall x, y, z \in A$ *if* $x \succeq y$ *and* $y \succeq z$ *then* $x \succeq z$*; and*
- *complete:* $\forall x, y \in A$ *either* $x \succeq y$ *or* $y \succeq x$ *or* $x \sim y$*.*

Proof. See Appendix A. □

Examples of Basic Preferences

Given $\mathcal{B} = \{0, 1\}$ and $BL = \mathcal{B}^2$, an example of basic preference defined according to Def. 10a is the following.

$$\begin{aligned}
(\langle sp_{100}, 0\rangle, \langle dist_1, 0\rangle) \succeq (\langle sp_{100}, 0\rangle, \langle dist_2, 0\rangle) \succeq \\
(\langle sp_{50}, 0\rangle, \langle dist_1, 0\rangle) \succeq (\langle sp_{50}, 0\rangle, \langle dist_2, 0\rangle) \succ \\
(\langle sp_{100}, 0\rangle, \langle dist_1, 1\rangle) \succeq (\langle sp_{100}, 1\rangle, \langle dist_1, 0\rangle) \succeq \\
(\langle sp_{100}, 0\rangle, \langle dist_2, 1\rangle) \succeq (\langle sp_{100}, 1\rangle, \langle dist_2, 0\rangle) \succeq \\
(\langle sp_{50}, 0\rangle, \langle dist_1, 1\rangle) \succeq (\langle sp_{50}, 1\rangle, \langle dist_1, 0\rangle) \succeq \\
(\langle sp_{50}, 0\rangle, \langle dist_2, 1\rangle) \succeq (\langle sp_{50}, 1\rangle, \langle dist_2, 0\rangle) \succ \\
(\langle sp_{100}, 1\rangle, \langle dist_1, 1\rangle) \succeq (\langle sp_{100}, 1\rangle, \langle dist_2, 1\rangle) \succeq \\
(\langle sp_{50}, 1\rangle, \langle dist_1, 1\rangle) \succeq (\langle sp_{50}, 1\rangle, \langle dist_2, 1\rangle)
\end{aligned} \tag{4.1}$$

Note that in preference (4.1), alternatives with lower required budget are preferred over alternatives with higher required budget and the agents' prefers sp_{100} over sp_{50} for every safety distance, and $dist_1$ over $dist_2$ for every speed.

Given $\mathcal{B} = \{0, 1\}$ and $BL = \mathcal{B}^2$, an example of basic preference defined according

to Def. 10b is the following.

$$\begin{aligned}
&(\langle sp_{100},0\rangle,\langle dist_1,0\rangle) \succ (\langle sp_{100},0\rangle,\langle dist_1,1\rangle) \succeq \\
&(\langle sp_{100},1\rangle,\langle dist_1,0\rangle) \succ (\langle sp_{100},1\rangle,\langle dist_1,1\rangle) \succ \\
&(\langle sp_{50},0\rangle,\langle dist_1,0\rangle) \succ (\langle sp_{50},0\rangle,\langle dist_1,1\rangle) \succeq \\
&(\langle sp_{50},1\rangle,\langle dist_1,0\rangle) \succ (\langle sp_{50},1\rangle,\langle dist_1,1\rangle) \succ \\
&(\langle sp_{100},0\rangle,\langle dist_2,0\rangle) \succ (\langle sp_{100},0\rangle,\langle dist_2,1\rangle) \succeq \\
&(\langle sp_{100},1\rangle,\langle dist_2,0\rangle) \succ (\langle sp_{100},1\rangle,\langle dist_2,1\rangle) \succ \\
&(\langle sp_{50},0\rangle,\langle dist_2,0\rangle) \succ (\langle sp_{50},0\rangle,\langle dist_2,1\rangle) \succeq \\
&(\langle sp_{50},1\rangle,\langle dist_2,0\rangle) \succ (\langle sp_{50},1\rangle,\langle dist_2,1\rangle)
\end{aligned} \tag{4.2}$$

Notice that in preference (4.2) states of affairs where sp_{100} and $dist_1$ hold are preferred over states of affairs where sp_{50} and $dist_1$ hold, regardless of the budget. Analogously, regardless of the budget, states of affairs where sp_{50} and $dist_1$ hold are preferred over states of affair where sp_{100} and $dist_2$ hold, which, in turn, are preferred over states of affair where sp_{50} and $dist_2$ hold. Such preference describes an agent type that prefers to drive fast rather than slow and that prefers to have a short safety distance rather than high, for whom maximizing speed and minimizing safety distance have priority over minimizing the budget to be spent, and, finally, who gives more importance to having a short safety distance rather than driving fast.

Finally, an example of a preference that does not satisfy Def. 10 is the following: $(\langle sp_{50},1\rangle,\langle dist_1,1\rangle) \succ (\langle sp_{50},0\rangle,\langle dist_1,0\rangle) \succ \ldots$. This is because the first two alternatives share the same propositional atoms but the alternative with higher required budget is preferred to the alternative with lower required budget.

Preferences

The basic preference as defined in Def. 10 may not be expressive enough to capture some realistic cases. In order to cover more cases and to make our approach applicable to model more realistic scenarios, we consider more complex types of agents' preferences that combines the two basic types of preferences (defined in Def. 10a and Def. 10b).

Intuitively, a rational agent may exhibit different preferences when the required budget increases. For example, consider a brave agent that prefers to drive fast and to keep a short safety distance rather than long, e.g., as per preference (4.2). Suppose, however, that such an agent is ready to pay only up to 1€ for driving fast and for keeping short safety distance. In such case, the agent would prefer to drive fast and to keep a short safety distance, compared to other alternatives (e.g., to drive slow and keep a long safety distance), if the required budget is lower than 1€. For example, in preference (4.2), ordered according to Def. 10b, we have $(\langle sp_{100},1\rangle,\langle dist_1,1\rangle) \succ (\langle sp_{50},0\rangle,\langle dist_1,0\rangle)$. If the required budget for either driving fast or keeping a short safety distance is higher than 1, however, the agent may instead give priority to spending the least possible. For example, $(\langle sp_{50},0\rangle,\langle dist_1,2\rangle)$, not reported in preference (4.2), would be preferred to $(\langle sp_{100},1\rangle,\langle dist_1,2\rangle)$, adhering

to Def. 10a instead of Def. 10b. In other words, a rational agent may use different criteria to order the alternatives in a preference depending on the required budget.

We formalize this intuition by defining a type of preference $(A, \succeq)$ that is a sequence of k basic preferences, with $1 \leqslant k \leqslant |\mathcal{B}|$. We call such a complex preference simply *preference*. Each of the k basic preferences adhere to either Def. 10a or Def. 10b, and the alternatives in the different basic preferences have *increasing* budgets. In particular, the set of possible budget lists $BL_i \subseteq \mathcal{B}^n$ for an alternative in the i-th basic preference $(A_i, \succeq_i)$, for $i \leqslant k$, is determined as per Def. 11.

Definition 11 (Budget Lists of the i-th Basic Preference)**.** *Consider a set $\mathcal{B} \subset \mathbb{N}$, and k disjoint subsets of $\mathcal{B}$, i.e., $\mathcal{B}_1, \ldots, \mathcal{B}_k$, such that each element of $\mathcal{B}_i$ is bigger than each element of $\mathcal{B}_j$, for $j < i \leqslant k$. In a preference composed by k basic preferences, the set of possible budget lists for the alternatives in the i-th basic preference $(A_i, \succeq_i)$, for $i \leqslant k$, is $BL_i = (\bigcup_{j \leqslant i} \mathcal{B}_j)^n \setminus \bigcup_{j<i} BL_j$*

For instance, given the set $\mathcal{B} = \{0, 1, 2\}$ and $k = 2$ two possible subsets of $\mathcal{B}$ as per Def. 11 are $\mathcal{B}_1 = \{0, 1\}$ and $\mathcal{B}_2 = \{2\}$. The possible budget lists for the alternatives of 2 basic preferences are therefore $BL_1 = \{(0,0), (0,1), (1,0), (1,1)\}$ and $BL_2 = \{(0,2), (1,2), (2,0), (2,1), (2,2)\}$. In other words, the budgets in the alternatives of the i-th basic preference are always lower or equal to $max(\mathcal{B}_i)$. This means that the required budget of every alternative in A_i is always *lower* or equal to $n \cdot max(\mathcal{B}_i)$, while the required budget of every alternative in A_{i+1} is always *higher* or equal to $n \cdot max(\mathcal{B}_i)$.

Definition 12 (Preference)**.** *Let $(A_1, \succeq_1), \ldots, (A_k, \succeq_k)$ be k basic preferences as per Def. 10, such that alternatives in A_i are defined with respect to a set of budget lists BL_i as per Def. 11. An agent is said to have a preference $(A, \succeq)$, iff $A = \bigcup_{i=1}^{k} A_i$ and $\succeq = \bigcup_{i=1}^{k} \succeq_i \cup \{(x, y) \mid x \in A_j \ \& \ y \in A_i \ \& \ 1 \leqslant j < i \leqslant k \}$.*

Note that a preference $(A, \succeq)$ that is composed by only one basic preference $(A_1, \succeq_1)$ so that $A = A_1$ for $BL_1 \subseteq \mathcal{B}^n$, and $\succeq = \succeq_1$, is a basic preference. If a preference is composed by more than one basic preference, every basic preference $(A_i, \succeq_i)$ composing the preference adheres to either Def. 10a or Def. 10b, and for every pair of alternatives $x, y \in A$ such that $x \in A_i, y \in A_j$ and $i < j$, it holds that $req\_bud(x) \leqslant req\_bud(y)$. Furthermore, notice that the sets $A_1, \ldots, A_k$ of alternatives of the k basic preferences composing a preference $(A, \succeq)$ are disjoint subsets of A, since the possible budget lists of the k basic preferences are disjoint subsets of $\mathcal{B}^n$.

Again, we note that a preference as per Def. 12 is transitive and complete.

Proposition 2. *A preference $Pref(a) = (A, \succeq)$ for an agent $a \in Ag$ is*

- *transitive: $\forall x, y, z \in A$ if $x \succeq y$ and $y \succeq z$ then $x \succeq z$; and*
- *complete: $\forall x, y \in A$ either $x \succeq y$ or $y \succeq x$ or $x \sim y$.*

Proof. See Appendix A. □

Examples of Preferences

An example of a preference composed by two basic preferences $(A_1, \succeq_1)$ and $(A_2, \succeq_2)$ is given in Eq. 4.3, given $\mathcal{B} = \{0, 1, 2\}$.

$$\begin{aligned}
&\#\textit{from here ordered by Def. 10b}\\
&(\langle sp_{100}, 0\rangle, \langle dist_1, 0\rangle) \succ (\langle sp_{100}, 0\rangle, \langle dist_1, 1\rangle) \succeq\\
&(\langle sp_{100}, 1\rangle, \langle dist_1, 0\rangle) \succ (\langle sp_{100}, 1\rangle, \langle dist_1, 1\rangle) \succ\\
&(\langle sp_{50}, 0\rangle, \langle dist_1, 0\rangle) \succ ... \succ (\langle sp_{50}, 1\rangle, \langle dist_1, 1\rangle) \succ ... \succ\\
&(\langle sp_{50}, 1\rangle, \langle dist_2, 1\rangle) \succ\\
&\#\textit{from here ordered by Def. 10a}\\
&(\langle sp_{100}, 0\rangle, \langle dist_1, 2\rangle) \succeq (\langle sp_{100}, 2\rangle, \langle dist_1, 0\rangle) \succeq ... \succ\\
&(\langle sp_{100}, 2\rangle, \langle dist_1, 2\rangle) \succ (\langle sp_{50}, 0\rangle, \langle dist_1, 2\rangle) \succeq ... \succeq\\
&(\langle sp_{50}, 2\rangle, \langle dist_2, 2\rangle)
\end{aligned} \tag{4.3}$$

In such preference, the budget lists of the alternatives in A_1 are elements of $BL_1 = \{(0,0),(0,1),(1,0),(1,1)\}$ for $\mathcal{B}_1 = \{0,1\}$, and the alternatives are ordered by Def. 10b. The budget lists of the alternatives in A_2, instead, are elements of the set $BL_2 = \{(0,2),(1,2),(2,0),(2,1),(2,2)\}$, for $\mathcal{B}_2 = \{2\}$, and they are ordered by Def. 10a. The required budget of every alternative in A_1 is lower or equal to 2, while the required budget of every alternative in A_2 is higher or equal to 2 and lower or equal to 4.

Consistent Preferences

The preferences above described allow to express a multitude of possible orderings between different states of affairs. In the following we define an additional property that a preference can exhibit. We call such property *consistency* [160].

Intuitively a preference is consistent if when a state of affairs where a propositional atom p holds is preferred to a state of affair where q holds, then states of affairs where p holds are preferred to states of affairs where q holds also when a third atom r is considered. For instance, if $(\langle p, b_1\rangle, \langle x, b_2\rangle) \succeq (\langle q, b_1\rangle, \langle x, b_2\rangle)$, then in a consistent preference this holds for every propositional atom x.

Notice that preferences as per Def. 12 are not necessarily consistent. An example of a preference that is not consistent (i.e., does no exhibit the consistency property) is the following:

$$\begin{aligned}
&(\langle sp_{100}, 0\rangle, \langle dist_1, 0\rangle) \succ (\langle sp_{80}, 0\rangle, \langle dist_1, 0\rangle) \succ (\langle sp_{80}, 0\rangle, \langle dist_2, 0\rangle) \succ\\
&(\langle sp_{50}, 0\rangle, \langle dist_1, 0\rangle) \succ (\langle sp_{50}, 0\rangle, \langle dist_2, 0\rangle) \succ (\langle sp_{100}, 0\rangle, \langle dist_2, 0\rangle) \succ ...
\end{aligned}$$

Notice that, given $dist_1$, sp_{100} is preferred to sp_{80}, but given $dist_2$, sp_{80} is preferred to sp_{100}.

We define consistent preferences by means of an *enumeration condition* over the propositional atoms of the alternatives. In particular, if two alternatives x and y with same budget lists differ exactly for one propositional atom, then if x is preferred to y, this has to hold also for all other pairs of alternatives with same budget lists differing exactly for the same propositional atoms as x and y. Intuitively the *enumeration*

condition imposes an ordering on the alternatives that corresponds to an ordering that can be obtained by systematically enumerating the possible combinations of propositional atoms. For instance, if, given $dist_1$, the proposition sp_{100} from the set $\{sp_{100}, sp_{50}\}$ is enumerated before proposition sp_{50} (i.e, $(\langle dist_1, b_1\rangle, \langle sp_{100}, b_2\rangle) \succ (\langle dist_1, b_1\rangle, \langle sp_{50}, b_2\rangle)$), then in a consistent preference sp_{100} is enumerated before sp_{50} also given $dist_2$ (i.e, $(\langle dist_2, b_1\rangle, \langle sp_{100}, b_2\rangle) \succ (\langle dist_2, b_1\rangle, \langle sp_{50}, b_2\rangle)$).

Definition 13. *A preference $Pref(a) = (A, \succeq)$ is consistent if and only if for all alternatives x, y in A s.t. their lists of propositional atoms differ exactly for one element, the following enumeration condition holds.*
Let $\diamond \in \{\succ, \sim\}$.
$x \diamond y \Rightarrow$
$\forall v, w \in A \mid prop(v) = (p_1, .., p_n)$ & $prop(w) = (p'_1, .., p'_n)$
if $p_i \neq p'_i$ & $\forall_{k \in \{1,\ldots,n\} \mid k \neq i} p_k = p'_k$ & $bud(v) = bud(w)$
then $v \diamond w$

4.3.4 Norms and Agents' Preferences

As mentioned above, in this chapter we assume that norms and agents' preferences are comparable. Consider $AL = (L_1, \ldots, L_n)$ and a norm set $\mathcal{N} = \langle N_1, \ldots, N_n\rangle$ as per Sec. 4.3.2. Given an alternative $(\langle p_1, b_1\rangle, ..., \langle p_n, b_n\rangle)$ in an agent's preference, we have that both the proposition p_i of i-th pair $\langle p_i, b_i\rangle$ and the proposition p of the i-th norm $N_i = (p, s)$ in $\mathcal{N}$ belong to L_i. Furthermore, since both the sanctions of the norms and the agents' budgets of agent's preferences are natural numbers, they also are commensurable. This makes it possible to analyze an agent's preference in the context of a norm to determine whether the preference motivates an agent to comply with a norm or to violate it.

Intuitively, in the context of a set of enforced norms, an agent that follows its preference aims at realizing a state of affairs that can be compliant with some of the enforced norms and violating other norms for which he is willing to pay the corresponding sanctions.

Given a set $\mathcal{N}$ of n norms and a preference $(A, \succeq)$, we say that an alternative $x \in A$ such that $x = (\langle p_1, b_1\rangle, \ldots, \langle p_i, b_i\rangle, \ldots, \langle p_n, b_n\rangle)$ is a *violating alternative* w.r.t. the i-th norm $N_i = (p, s)$ in $\mathcal{N}$, and we write $viol(x, N_i)$, if and only if p_i (e.g., sp_{100}) excludes[3] p (e.g., sp_{50}); otherwise x is said to be a *complying alternative* w.r.t. norm N_i. An alternative that is compliant w.r.t. all norms in $\mathcal{N}$ is said *fully compliant*. Note that any rational preference, due to its completeness property as per Prop. 2, always contains at least one fully compliant alternative. This means that agents always have a choice to aim at a state of affairs that does not violate any norm.

Definition 14 (Most Preferred Alternatives to Act Upon)**.** *Given a preference $(A, \succeq)$ and a set $\mathcal{N}$ of n norms, a subset $A' \subseteq A$ of alternatives is called the set of most preferred alternatives to act upon in the context of $\mathcal{N}$ if and only if for all alternatives*

[3] In this chapter, we assume that information about exclusion between propositional atoms (e.g., in the sense of material implication) is given as background knowledge. A formal definition of violation of a norm depends on the specific language used to specify the norms and is out of the scope of the chapter.

$x \in A \backslash A'$ it holds that for all alternatives $y \in A'$ either $y \succ x$ or $x \succeq y$ and there exists a norm $N_j = (p, s)$ in $\mathcal{N}$ s.t. $viol(x, N_j)$ & $b_j < s$ (with b_j budget of the j-th pair in x).

The set of most preferred alternatives to act upon in the context of $\mathcal{N}$ is the set of alternatives $A' \subseteq A$ such that every other alternative $x \in A \backslash A'$ is either strictly less preferred (i.e., $y \succ x \ \forall y \in A'$), or is an alternative that violates at least a norm N_j but the budget is not enough to pay the sanction (i.e., $viol(x, N_j)$ & $b_j < s$). This means that the alternatives in A' are either fully compliant or they violate some norms and the budget is enough to pay the sanction, and there is no other alternative that satisfies such conditions that is strictly preferred to them.

A rational agent always acts upon one of its most preferred alternatives. We say that an agent a has *a reason to violate* a norm N whenever the agent's preference $Pref(a)$ is so that, among the set of most preferred alternatives, there is at least one alternative x such that $viol(x, N)$. When different alternatives are equally preferred by an agent, the agent can freely choose to aim at any of them. This means that an agent that has a reason to violate a norm will not necessary aim to violate it: if another alternative is equally preferred to the violating state of affairs, the agent may decide to aim to the obeying state of affairs, despite it has a reason to violate the norm. Consider for example an agent type characterized by the preference in Eq. 4.1 and a norm $N = (sp_{50}, 0)$ that prohibits agents to drive faster than $50 km/h$. Given N, the agents' most preferred alternatives to act upon are $(\langle sp_{100}, 0\rangle, \langle dist_1, 0\rangle)$, $(\langle sp_{100}, 0\rangle, \langle dist_2, 0\rangle)$, $(\langle sp_{50}, 0\rangle, \langle dist_1, 0\rangle)$ and $(\langle sp_{50}, 0\rangle, \langle dist_2, 0\rangle)$. Some of these alternatives violate the norm N (e.g., $(\langle sp_{100}, 0\rangle, \langle dist_1, 0\rangle)$), therefore the agent has a reason to violate N. However, some of the other most preferred alternatives are compliant with the norm (e.g., $(\langle sp_{50}, 0\rangle, \langle dist_2, 0\rangle)$). Since all most preferred alternatives are equally preferred, the agent may rationally decide to aim at any of them.

We introduce the notion of *maximum budget for norm violation* as the maximal payment that an agent is willing to pay for violating a given norm according to its preference. Let $N_i = (p, s)$ be the i-th norm in $\mathcal{N}$, and let $Pref(a) = (A, \succeq)$ be the preference of agent a. Let $x \in A$ be the agent's most preferred *fully compliant* alternative, and $A' = \{y \in A \mid y \succeq x\}$ be the set of alternatives in A that are (equally) preferred to x. The maximum budget that a is willing to pay for the violation of N_i, denoted as $maxB(a, N_i)$, is the highest budget b that occurs in the i-th pair of the alternatives in A'. Note that if the maximum budget for violating a norm is lower than the sanction of norm N_i, then the most preferred alternatives to act upon are necessarily alternatives compliant w.r.t. N_i. For instance if $N = (sp_{50}, 3)$ and an agent a has $maxB(a, N) = 2$, then all alternatives x in the set of most preferred alternatives are compliant to N, i.e., $viol(x, N)$ does not hold, and it does not exists a pair $\langle p, b\rangle \in x$ with $b \geqslant 3$, since $b \leqslant maxB(a, N) < 3$.

Finally, it is worth noting that in case of preference composed by more than one basic preference as per Def. 12, it is always the case that if the first basic preference is strictly preferred to the remaining ones then the set of most preferred alternatives to act upon in the context of $\mathcal{N}$ never contains any alternatives from any basic preference apart from the first one. This is because the first basic preference necessarily contains an alternative that is fully compliant (due to completeness of every basic preference $(A_i, \succeq_i)$ w.r.t. AL and BL_i for $1 \leqslant i \leqslant k$ and k number of basic preferences composing

the preference), and such alternative is strictly preferred to any other alternative that belongs to the remaining basic preferences.

4.4 Norm-based Supervision

In this section, we present the key concepts of a norm-based supervision of a multi-agent system. We build on the run-time norm-based supervision mechanism for multi-agent systems as proposed in [117] and sketched in Fig. 4.2. Such mechanism corresponds to a control loop that continuously monitors the behavior of a multi-agent system, evaluates the enforcement of the norms w.r.t. the system-level objectives, and, when needed, intervenes by revising the norms.

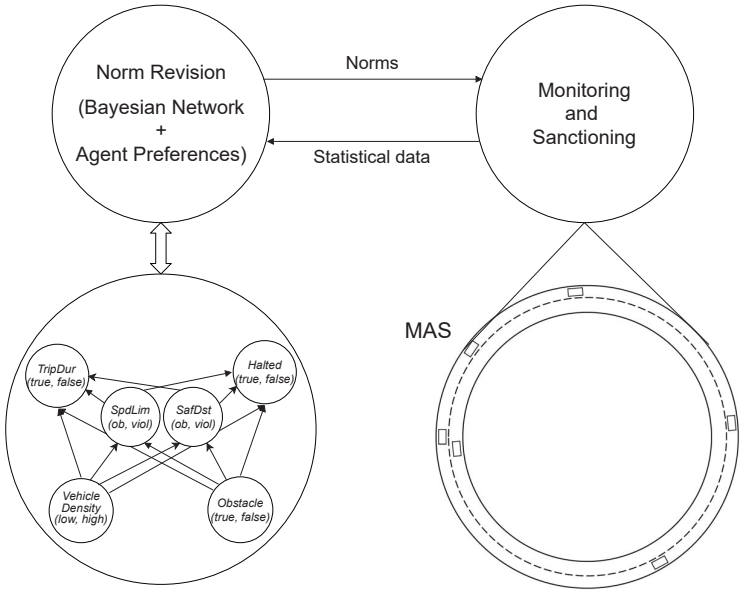


Figure 4.2: *Illustration of the MAS supervision mechanism.*

Consider an ordered set $\mathcal{N} = \langle N_1, ..., N_n \rangle$ of norms and a set $\mathcal{C}$ of all possible operating contexts of the multi-agent system (e.g, a context $c \in \mathcal{C}$ in the ring road scenario could be "low vehicle density and no obstacle"). We call *system configuration* an assignment of a sanction $s \in \mathbb{N}$ to each norms in $\mathcal{N}$ in each of the MAS operating contexts.

For example, given two possible operating contexts c_1 and c_2, and given a norm set $\mathcal{N} = \{N_1, N_2\}$, a possible system configuration is $\{(c_1, (N_1, 1), (N_2, 0)), (c_2, (N_1, 0), (N_2, 1))\}$, meaning that in context c_1 norms N_1 and N_2 are enforced respectively with sanctions 1 and 0, while in context c_2 they are enforced respectively with sanctions 0 and 1.

The control loop of the supervision mechanism sketched in Fig. 4.2 starts with an initial system configuration. A *Monitoring and Sanctioning* component collects, at run-time, perfect information about the obedience or violation of the norms in the

contexts in which they are evaluated and sanctions agents that violate the norms. Such component also provides a Boolean evaluation of the system-level objectives (e.g., whether the number of halted cars is below a certain threshold or not, in the ring road scenario).

The collected information is used to automatically train a Bayesian Network called *Norm Bayesian Network* (described in Sec. 4.4.1) that is used to learn and reason at run-time about the correlation between norm obedience or violation and the achievement of the system-level objectives. For example, the *Norm Bayesian Network* helps answering questions like how well, and in which contexts, does the norm $(sp_{50}, 100)$ help achieve the objective of avoiding halted cars?

A *Norm Revision* component makes use of the learned knowledge, encoded in the Bayesian Network, to determine whether some norms should be revised and how. Revising a norm $N = (p, s)$ means modifying either the proposition p or the sanction s, or both. In this chapter, we focus on the revision of the sanctions of the norms. The norm revision process generates as output a (possibly) new system configuration, replacing the current one.

In previous work [117], we proposed an implementation of the control loop described above as a variation of the hill climbing optimization technique. In this chapter, we follow the same approach. We consider the system configurations as possible solutions to explore in order to find an optimal one. The quality of a solution is determined, by means of the observed data from MAS execution, as the probability of achieving the system-level objectives. Instead of terminating the exploration of the space when a local optimum is found, as in traditional hill climbing, we use as stopping criterion a constraint defined by the system designer that determines whether or not the current solution is acceptable. In particular we use, as stopping criterion, a minimum desired value of the probability of achieving the system-level objectives. We call such value t_{oa}. We use the *Norm Revision* component to determine the next solution to try, when the current one is not acceptable.

In [117] we proposed heuristic algorithms for suggesting norm revisions that alter the *regimented norms*. In this chapter, differently from the earlier work, we make use of some additional information concerning the preferences of the agents in order to determine how to revise the norms, and we focus on the revision of sanctions. In [119], we used the same framework of [117] to revise the way one norm is *enforced* by modifying its sanction. In this chapter, we significantly extend our previous work by devising several new strategies for the revision of the sanctions of *multiple norms* enforced at the same time.

In the rest of the section we first provide some background concerning the *Norm Bayesian Network*, then we analyze some properties of the relationships between norms, agents' preferences and system-level objectives.

4.4.1 Norm Bayesian Network

Consider some monitorable environmental properties such as the density of vehicles or the presence of an obstacle on the ring road. Each of these properties is called *contextual variable*, and is associated to a domain of values. For example, *Vehicles density* can be either *low* or *high*, while *Obstacle* can be *true* or *false*. Given a set

of contextual variables, a *context* assigns a value to each contextual variable. For instance, given *Vehicles density* and *Obstacle*, four possible contexts exist: *high-true*, *high-false*, *low-true*, *low-false*.

A *Norm Bayesian Network* $\mathcal{NBN} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$ [117] is a Bayesian Network where:

- $\mathcal{X} = \mathbf{N} \cup \mathbf{O} \cup \mathbf{C}$ are nodes that represent random variables in probability theory. $\mathbf{N}$, $\mathbf{O}$ and $\mathbf{C}$ are disjoint sets. $\mathbf{N}$ consists of *norm nodes*; each node $N \in \mathbf{N}$ corrensponds to a norm and has a discrete domain of 3 possible values: *obeyed*, *violated* and *disabled*. $\mathbf{O}$ consists of *objective nodes*; each node $O \in \mathbf{O}$ corresponds to a Boolean objective and has a discrete domain of 2 values: *true* and *false*. Finally, $\mathbf{C}$ consists of *context nodes*; each node $C \in \mathbf{C}$ corresponds to a contextual variable and can have a discrete or continuous domain of values.

- $\mathcal{A} \subseteq (\mathbf{C} \times \mathbf{N}) \cup (\mathbf{C} \times \mathbf{O}) \cup (\mathbf{N} \times \mathbf{O})$ is the set of arrows that connect pairs of nodes. If there is an arrow from node X to node Y, X is called parent of Y.

- $\mathcal{P}$ is a set of conditional probability distributions. These are encoded into conditional probability tables (CPTs), each one associated with a node in $\mathcal{X}$ and quantifying the effect of the parents on the node. The conditional probability values in the CPT of a node are the *parameters* of the network. These parameters are automatically learned from observed data from MAS execution through classic Bayesian learning.

Notation. In the rest of the chapter, we use the following notation for Bayesian Networks. Italic uppercase (X, Y, ...) for random variables; bold uppercase ($\mathbf{X}$, $\mathbf{Y}$, ...) for sets of random variables; italic lowercase (v_1, v_2, ...) for values in the domain of a random variable; N_v abbreviates ($N = v$), i.e., an assignment of value v to a norm variable N; $\mathbf{O}_v$ denotes an assignment of value v to all nodes in $\mathbf{O}$; P denotes a single probability. An evidence $\mathbf{e}$ is an observed assignment of values for some or all of the random variables in the network. An evidence $\mathbf{c}$ for all the context nodes $\mathbf{C}$ is an observation for a certain context; for example, *Vehicles density* has value *low* and *Obstacle* has value *false*. For simplicity, we use the term *context* also to refer to the associated evidence in the Bayesian Network.

Fig. 4.3 reports an example of a *Norm Bayesian Network* for the running example of the ring road.

Since we focus on revising the sanctions that enforce norms, norms are never disabled, therefore in the following we ignore the *disabled* value of the nodes in the Bayesian Network. Despite we do not explicitly disable a norm, we consider enforcing a norm with a sanction of 0 as equivalent to disabling the norm, assuming that an agent that violates a norm with sanction of 0 does not incur in any other kind of sanctions (e.g., consequences in the relation between the individual and the other agents due to shared (moral) values [31]).

Finally, the construction and training of the *Norm Bayesian Network* is a fully automated process. In particular, the structure of the network can be trivially obtained from the the definition of $\mathcal{X}$ and $\mathcal{A}$. The conditional probability distributions $\mathcal{P}$ (i.e., the parameters of the network), instead, are automatically learned through classical Bayesian learning using data collected from MAS execution. Without going

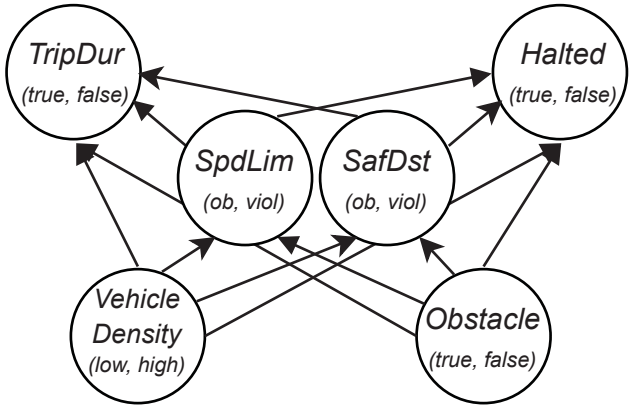


Figure 4.3: *A Norm Bayesian Network for the ring road.*

into the details of the *Monitoring and Sanctioning* component, which are out of the scope of this chapter, Table 4.1 reports a sample dataset that can be obtained from monitoring norms and objectives for the running example of the ring road. The values that each of the variables assumes belongs to its domain as above specified (e.g., *obeyed*, *violated*, for norm nodes, *true* or *false* for objective nodes). Such dataset can be used to automatically train the *Norm Bayesian Network* of Fig. 4.3 and learn the set of conditional probability distributions $\mathcal{P}$. As in this work we assume that the population of agents do not change over time and that the behavior of agents is consistent over time, the CPTs of the *Norm Bayesian Network* stabilize after receiving a sufficient number of evidences.

Table 4.1: *Example of part of a dataset used to train the Norm Bayesian Network of Fig. 4.3 and obtained from monitoring the execution of the MAS.*

VehicleDensity	*Obstacle*	*SpdLim*	*SafDst*	*TripDur*	*Halted*
low	true	viol	ob	true	false
low	false	ob	viol	false	false
high	true	viol	ob	true	false
high	false	ob	ob	true	true
...					

4.4.2 Norms, Agents' Preferences and System-level Objectives

Consider a set of agent types $\mathcal{T} = \{t_1, \ldots, t_k\}$, each type corresponding to a preference as per Sec. 4.3. In order to focus on the revision of the norms' sanctions, we assume that we possess a correct estimation of the preferences of agents concerning the aspects of the system we aim to regulate. Additionally, we assume that the agents' preferences do not change in different contexts. As we will see in the following, an accurate estimation of agents' preferences is helpful for improving the effectiveness of

our heuristics. Our technique, however can be extended to support partial or inaccurate estimations of the agents' preferences. In Section 4.7.1, we sketch some directions for future work to support these aspects.

Take a set of agents $Ag = \{a_1, \ldots, a_n\}$, each with a specific type from $\mathcal{T}$. We use $Pref(a) \in \mathcal{T}$ to indicate that agent $a \in Ag$ behaves according to a type from $\mathcal{T}$. For simplicity we assume that the behaviors exhibited in the multi-agent system are uniformly distributed over all the agents: at every time instant every agent either violates or obeys each of the enforced norms.

Given these assumptions and a set of norms $\mathcal{N}$, we say that a norm N in $\mathcal{N}$ is *well defined* in the context of $\mathcal{N}$ (simply *well defined*, for brevity) if the probability that N is violated, denoted as $P(N_{viol})$, is never higher than the percentage of agents in the MAS with a reason to violate N in the context of $\mathcal{N}$.[4] In other words, the *upper bound* of the probability $P(N_{viol})$ in the context of $\mathcal{N}$ (denoted as $UB(N_{viol}, \mathcal{N})$) is the percentage of the agents with a reason to violate N in the context of $\mathcal{N}$.

Let N be a norm in $\mathcal{N}$, and let $\delta = (d_1, ..., d_k)$ be a distribution over the agent types $\mathcal{T} = \{t_1, \ldots, t_k\}$, where $d_i \in [0, 1]$ is the percentage of population of agents of type t_i, with $\sum_{i=1}^{k} d_i = 1$. The percentage of agents with a reason to violate N (as per Sec. 4.3.4) in the context of $\mathcal{N}$ is $\sum_{i=1}^{k}(d_i \cdot hasReason(i, N, \mathcal{N}))$, with $hasReason(i, N, \mathcal{N}) = 1$ if agent type t_i has a reason to violate N in the context of $\mathcal{N}$, *0* otherwise.

Consider, as an example, a norm set $\mathcal{N} = \langle N_1, N_2 \rangle$, with $N_1 = (sp_{50}, s_1)$ and $N_2 = (dist_2, s_2)$ and $\mathcal{B} = \{0, 1\}$. Consider the two types of agents t_1 and t_2 as per Eq. 4.1 and Eq. 4.2, respectively. Assuming a uniform distribution of agents between the two types, Fig. 4.4 reports the upper bound of the probability of violating N_1 and N_2 for this example with different sanctions (i.e., different values of s_1 and s_2).

The upper bound of $P(N_{viol})$ describes a *worst-case* hypothetical situation where all agents behave according to their preferences, and if they have reason to violate a norm they are assumed to violate it, no contextual factor influences agent behavior, and interactions among agents do not prevent them to act according to their preferences. This would happen, for example, when a single car drives on an empty highway with perfect road and car conditions. Note, however, that the actual probability to violate a norm is affected by the agents' decisions, their interactions and by the MAS environment, and it is assumed to be unknown *a priori*. Even if all agents have a reason to violate a norm, due to their interaction or to environmental circumstances (e.g., large number of cars on the ring-road), none of them may end up violating it. Furthermore, as explained in Sec. 4.3.4, if an agent equally prefers two states of affairs, one violating a norm, and another obeying the norm, the agent, since autonomous, may decide to obey the norm even if it has a reason to violate it. We

[4] Consider a norm N = *every vehicle on the ring road shall always exceed 70km/h*, a type of norm employed in our society, for instance, to prevent vehicles to have negative impact on road throughput and safety. Our framework supports such type of norm if it is well-defined. Suppose that in our running example no agent has reason to violate N. If N was well-defined we would expect $P(N_{viol}) = 0$. However, *in our running example*, such norm is not well-defined, for in case of high density, for example, the agents may be forced to slow down below the minimum speed, therefore violating the norm and exhibiting $P(N_{viol}) > 0$. A well-defined norm guarantees agents that have no reason to violate the norm (i.e., their preferred alternatives are compliant with the norm) to be able to obey such norm.

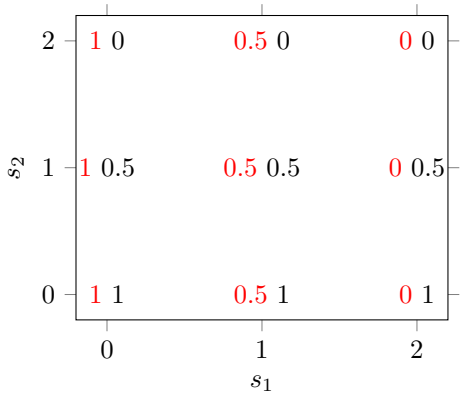


Figure 4.4: *Upper bound of the probability of violating norms $N_1 = (sp_{50}, s_1)$ (in red) and $N_2 = (dist_2, s_2)$ (in black) with the two types of agents t_1 and t_2 as per Eq. 4.1 and Eq. 4.2, respectively, uniformly distributed.*

call, therefore, the monitored probability of violating (obeying) a norm *exhibited norm violation* (*obedience*). We do not assume any prior knowledge about such probability.

Note that, since we consider agent types with rational preferences as per Sec. 4.3.3, increasing the sanction s of a norm $N = (p, s)$, without changing the sanctions of other norms, does not increase the percentage of agents with a reason to violate N. Therefore, given k agent types and $maxB(\mathcal{T}, N)$ as the maximum budget among all agent types to violate a well-defined norm $N = (p, s)$, the percentage of agents with a reason to violate a well-defined norm $N' = (p, maxB(\mathcal{T}, N) + 1)$ in the context of $\mathcal{N}$ is 0. This is to say that increasing the sanction of a norm above the maximum budget that any agent is willing to pay causes all agents to comply with the norm. Consequently, given two well-defined norms $N = (p, s_1)$ and $N' = (p, s_2)$ such that $s_2 > s_1$, and assuming no change in other norms of $\mathcal{N}$, the upper bound of the probability $P(N'_{viol})$ is never bigger than the upper bound of the probability $P(N_{viol})$.

Furthermore, it is possible to prove that, if all agents in the MAS have a consistent preference (as per Def. 13), then given a set of norms $\mathcal{N} = \langle N_1, ..., N_n \rangle$, increasing the sanction of a norm N_j in $\mathcal{N}$ without changing the sanctions of other norms, does not increase the upper bound of the probability $P(N_{viol})$ for every N in $\mathcal{N}$.

Proposition 3. *Given an ordered set of norms $\mathcal{N} = \langle N_1, ..., N_n \rangle$, and a set of t agent types $\mathcal{T}$, each type corresponding to a consistent preference (as per Def. 13), increasing the sanction of a norm N_j in $\mathcal{N}$ without changing the sanctions of other norms, does not increase the upper bound of the probability $P(N_{viol})$, i.e., $UB(N_{viol}, \mathcal{N})$, for all N in $\mathcal{N}$.*

Proof. See Appendix A. □

The concept of well-defined norm as described above, concerns the relationship between a norm and the preferences of the agents. In a multi-agent system, norms are enforced in order to achieve some system-level objectives. Although setting the sanction of all norms in $\mathcal{N}$ above $max(\mathcal{B})$ makes all the agents fully compliant (i.e.,

$P(N_{viol}) = 0$ and $P(N_{ob}) = 1$ for all $N \in \mathbf{N}$), this does not necessarily guarantee the achievement of the system-level objectives, as norms can be ineffective, or even harmful, when obeyed by all agents [117]. Having an estimation of the agents' preferences on its own is therefore not sufficient for an effective supervision of a MAS.

We describe here two properties that, instead, relate a norm with the system-level objectives: the concept of *synergy* between a norm and the system-level objectives, and the concept of *effectiveness* of a norm set.

We say that there is a *positive synergy* between a norm and the system-level objectives if it is more likely to achieve the system-level objectives when the norm is obeyed than when it is violated. A *positive synergy* between a norm N and a set of Boolean objectives $\mathbf{O}$ exists if $P(\mathbf{O}_{true}|N_{ob}) > P(\mathbf{O}_{true}|N_{viol})$. We say that there is a *negative synergy* between N and $\mathbf{O}$ if $P(\mathbf{O}_{true}|N_{ob}) < P(\mathbf{O}_{true}|N_{viol})$. Finally, we say that there is *no synergy* between N and $\mathbf{O}$ if $P(\mathbf{O}_{true}|N_{ob}) = P(\mathbf{O}_{true}|N_{viol})$.

We say, instead, that a norm set $\mathcal{N}$ is *effective* if, when norms in $\mathcal{N}$ are enforced, $\mathcal{N}$ guarantees the desired achievement level t_{oa} of the system-level objectives, i.e., when $P(\mathbf{O}_{true}) \geqslant t_{oa}$. Conversely, if, when enforcing a norm set $\mathcal{N}$, we have that $P(\mathbf{O}_{true}) < t_{oa}$, we say that $\mathcal{N}$ is *ineffective*.

Information such as the exhibited norm obedience, the synergy and the effectiveness described above, are hard to determine while designing a MAS. This is due to several factors, including the complexity of the system, the interaction between autonomous agents, the lack of complete knowledge of the agents' internals, and the uncertainty of the environment. However, they can be learned at run-time by monitoring the MAS execution. In this chapter, we learn such properties by means of the *Norm Bayesian Network* and, in Sec. 4.5, we propose different strategies to combine these properties with the agents' preferences, in order to revise the sanctions of an ineffective norm set $\mathcal{N}$.

4.5 Norm Revision

In this section, we propose different heuristic strategies for the revision of the sanctions of a set of norms whose enforcement is currently ineffective (as per Sec. 4.4.2). Opportune sanctioning of agents is a well-known mechanism to achieve the system-level objectives in MASs [68, 72]. Our strategies leverage the knowledge learned at run-time about norm effectiveness and an estimation of the preferences of the agents in the system, and determine a new set of sanctions to use to enforce the norms.

Take the *Norm Bayesian Network* in Fig. 4.3. By analyzing the CPTs of the objectives nodes $\mathbf{O} = \{TripDur, Halted\}$, we can determine whether a norm set $\mathcal{N}$ is effective or not in a context $\mathbf{c}$. If $\mathcal{N}$ is not effective (i.e., $P(\mathbf{O}_{true}|\mathbf{c}) < t_{oa}$), a norm revision process is triggered. In such case, in this chapter we aim to revise the sanctions of the norms in $\mathcal{N}$. For example, if the two norms $(sp_{50}, 1)$ and $(dist_1, 1)$ are ineffective when on the ring road there is an obstacle and high vehicle density, we aim to identify another set of values for their sanctions. Given a norm set $\mathcal{N}$ consisting of n norms, a set of agent types $\mathcal{T}$ and the maximum possible budget $max(\mathcal{B})$ among all agent types in $\mathcal{T}$, the possible sets of sanctions that can be used to enforce norms in

$\mathcal{N}$ is $\mathcal{S} = \times_{i=1}^{n}\{s \in \mathbb{N} \mid s \leqslant max(\mathcal{B}) + 1\}$. When a norm is enforced with a sanction 0, agent's decisions are not affected by the norm, since every agent can always afford to violate (if preferred) a norm with sanction 0. When a norm is enforced with a sanction $max(\mathcal{B}) + 1$, instead, no agent can violate such norm, since no agent can afford to pay such sanction, for the maximum possible budget among all agent types is $max(\mathcal{B})$.

The set $\mathcal{S}$ is the search space within which our heuristic strategies for norm revision search for new sanctions.

In Sec. 4.5.1, we describe six strategies for the suggestion of a revision of the sanctions of a norm set. Such strategies extend and adapt heuristics presented in previous work [117, 119] by supporting the revision of sanctions of *multiple norms*. Each strategy suggests how the behavior of agents w.r.t. the aspects of the system regulated by norms should change in order to improve the probability of achieving the system-level objectives. For example, given two norms, one strategy could suggest to reduce the violations of one norm and to increase the violations of the second norm. Based on the upper bound of the violation of norms obtained from agents' preferences (Sec. 4.4.2), we provide then in Sec. 4.5.2 an algorithm to explore the search space $\mathcal{S}$ in order to identify a new set of sanctions that satisfies (as much as possible) the suggestions provided by the revision strategies.

It is worth noting that we do not claim that modifying sanctions is always enough in order to achieve the system's objectives. As shown in previous work [117], sometimes the enforced norms (and not their enforcement) need to be revised. In this chapter, however, we focus on mechanisms for the revision of the sanctions associated to the norms (i.e., the way norms are enforced). The combination of the mechanisms proposed here with the revision of the content of the norms is left for future work.

4.5.1 Norm Revision Strategies

We propose six strategies for the suggestion of norm revisions. Each strategy determines a list of n suggestions (one per each norm in $\mathcal{N}$). We present three types of strategies: *synergy-based strategies*, *sensitivity-based strategies*, and *category-based strategies*.

Each strategy is applied to a context **mpc** that, in our framework, corresponds to the most problematic context in which the objectives are not achieved. In particular, $\mathbf{mpc} = argmax_{\mathbf{c} \in all(\mathbf{c})} P(\mathbf{O}_{false} \mid \mathbf{c})$, where $all(\mathbf{c})$ is the set of all possible contexts (assignments of a value to each of the context nodes in $\mathcal{NBN}$). For simplicity, in the rest of the section, we call such context simply $\mathbf{c}$.

Synergy-based strategies

Synergy-based strategies are based on the concept of norm-objectives synergy described in Sec. 4.4.2. The idea is that, if there is a positive synergy between a norm N and the objectives $\mathbf{O}$ in $\mathbf{c}$, the objectives $\mathbf{O}$ are more likely to be achieved when N is obeyed. In this case, by reducing the violations of N, we expect to increase $P(\mathbf{O}_{true}|\mathbf{c})$. If there is a negative synergy between N and $\mathbf{O}$ in $\mathbf{c}$, instead, we expect that increasing the violations of N, and $P(\mathbf{O}_{true}|\mathbf{c})$ would increase. We present two

strategies of this type (*Naive synergy* and *Combined synergy*), which differ in the way they determine the synergy between norms and objectives.

Naive synergy. Consider, *for each norm* $N \in \mathbf{N}$, its synergy with the objectives $\mathbf{O}$:

$$argmax_{v \in \{ob, viol\}} P(\mathbf{O}_{true} | N_v \wedge \mathbf{c}) \tag{4.4}$$

For instance, for a norm node *SpdLim* in the Bayesian Network of Fig. 4.3, where $\mathbf{O} = \{TripDur, Halted\}$, we have that

$$P(\mathbf{O}_{true} \mid N_v \wedge \mathbf{c}) = P(TripDur_{true}, Halted_{true} \mid SpdLim_v \wedge \mathbf{c} \wedge SafDst_{ob}) \cdot P(SafDst_{ob} \mid \mathbf{c}) + P(TripDur_{true}, Halted_{true} \mid SpdLim_v \wedge \mathbf{c} \wedge SafDst_{viol}) \cdot P(SafDst_{viol} \mid \mathbf{c})$$

To determine the *argmax* of Eq. 4.4 means therefore to determine if $SpdLim_{ob}$ is better than $SpdLim_{viol}$ for the achievement of the objectives *TripDur* and *Halted*.

Naive synergy calculates such *argmax* for each norm node and suggests to decrease violations of norms such that $v = ob$ in Eq. 4.4, and to increase violations of norms where $v = viol$ in Eq. 4.4. For instance, given $\mathcal{N} = \langle N1, N2 \rangle$, if $v = ob$ for $N1$ and $v = viol$ for $N2$, then *naive synergy* suggests to decrease violations of norm $N1$ and to increase violations of norm $N2$.

Combined synergy. Determine which combination of values *obeyed* and *violated* for each norm is the best for the achievement of the objectives $\mathbf{O}$.
Let $\mathbf{ov}$ be the set of all possible assignments of values in the set $\{ob, viol\}$ to all norm nodes in $\mathbf{N}$ (e.g., given $\mathbf{N} = \{N1, N2\}$, then $\mathbf{ov} = \{\{N1_{ob}, N2_{ob}\}, \{N1_{ob}, N2_{viol}\}, \{N1_{viol}, N2_{ob}\}, \{N1_{viol}, N2_{viol}\}\}$). Determine:

$$\mathbf{n}_d = argmax_{\mathbf{n} \in \mathbf{ov}} P(\mathbf{O}_{true} | \mathbf{n} \wedge \mathbf{c}) \tag{4.5}$$

This strategy suggests to decrease violations of norms with value *ob* in $\mathbf{n}_d$ and to increase violations of norms with value *viol* in $\mathbf{n}_d$. For instance if $\mathbf{n}_d = \{N1_{ob}, N2_{viol}\}$, then *combined synergy* suggests to increase violations of norm $N1$ and to decrease violations of norm $N2$.

It is worth noting that *Combined synergy* purely determines the best combination of values for the norms, according to the observed data from MAS execution, without considering the prior probability of observing those values (in practice, *Combined synergy* only compares, one by one, the rows of the CPT of the objective nodes). *Naive synergy*, instead, when comparing different combinations of values for the norms, takes also into account the probability to observe those values (*Naive synergy* compares sums of different rows of the CPT of the objectives nodes, multiplied by the prior probability of observing the corresponding values for the norm nodes). Adopting the *Naive synergy* strategy may have the advantage of providing more precise suggestion w.r.t. the data acquired so far during the system execution. Considering only the CPT of the objective nodes, as per *Combined synergy*, may help instead determining the actual best combination of values of obedience of the norms for the system-level objectives, without being biased by the current probabilities of violating the norms, which will be modified after the sanctions revision.

Sensitivity-based strategies

Sensitivity-based strategies are based on the sensitivity analysis technique from probabilistic reasoning [76]. Such strategies do not only determine the direction of the revision—i.e., increasing or decreasing the probability of violating a norm, as in the case of synergy-based strategies—, but also estimate the required change in such probability in order to make the entire norm set effective in context $\mathbf{c}$. In particular, given a norm node N, the probability $P(N_{viol}|\mathbf{c})$ is a parameter $\theta_{N_{viol|\mathbf{c}}}$of the *Norm Bayesian Network*. Sensitivity-based strategies try to identify possible changes to the parameter $\theta_{N_{viol|\mathbf{c}}}$that can ensure the satisfaction of the constraint $P(\mathbf{O}_{true}|\mathbf{c}) \geqslant t_{oa}$. We call *required revision strength* (*RRS*) for a norm set $\mathcal{N} = \langle N1, ...Nn\rangle$, the set of desired changes $\{\Delta\theta_{N1_{viol|\mathbf{c}}}, ..., \Delta\theta_{Nn_{viol|\mathbf{c}}}\}$ in the parameters $\theta_{N_{viol|\mathbf{c}}}$of each N in $\mathcal{N}$ that ensure the satisfaction of the constraint $P(\mathbf{O}_{true}|\mathbf{c}) \geqslant t_{oa}$. We present two strategies of this type (*Naive sensitivity analysis* and *n-CPT sensitivity analysis*), which differ in the way they determine such set of desired changes for each norm in $\mathcal{N}$.

Naive sensitivity analysis. Determine, *for each norm* N, the required revision strength (*RRS*) $\Delta\theta_{N_{viol}}$ by solving equation 4.6.

$$P(\mathbf{O}_{true}|\mathbf{c}) + \frac{\delta P(\mathbf{O}_{true}|\mathbf{c})}{\delta\theta_{N_{viol|\mathbf{c}}}} \cdot \Delta\theta_{N_{viol|\mathbf{c}}} \geqslant t_{oa} \tag{4.6}$$

Consider the topology of a *Norm Bayesian Network*. Following Chan *et al.* [76], the derivative $\frac{\delta P(\mathbf{O}_{true}|\mathbf{c})}{\delta\theta_{N_{viol|\mathbf{c}}}}$ for a norm node N in $\mathcal{N}$ can be computed as follows.

$$\frac{\delta P(\mathbf{O}_{true}|\mathbf{c})}{\delta\theta_{N_{viol|\mathbf{c}}}} = \frac{P(\mathbf{O}_{true}, N_{viol}|\mathbf{c})}{P(N_{viol}|\mathbf{c})} - P(\mathbf{O}_{true}|N_{ob}, \mathbf{c}) \tag{4.7}$$

For instance, for a norm node *SpdLim* in the Bayesian Network of Fig. 4.3, where $\mathbf{O} = \{TripDur, Halted\}$, the left member of the difference in Eq. 4.7 is

$$\frac{P(\mathbf{O}_{true}, N_{viol}|\mathbf{c})}{P(N_{viol}|\mathbf{c})} = \frac{P(TripDur_{true}, Halted_{true}, SpdLim_{viol}|\mathbf{c})}{P(SpdLim_{viol}|\mathbf{c})} =$$
$$= \frac{P(TripDur_{true}, Halted_{true}|SpdLim_{viol}, \mathbf{c}) \cdot P(SpdLim_{viol}|\mathbf{c})}{P(SpdLim_{viol}|\mathbf{c})} =$$
$$P(TripDur_{true}|SpdLim_{viol}, \mathbf{c}) \cdot P(Halted_{true}|SpdLim_{viol}, \mathbf{c}) =$$
$$= P(TripDur_{true} \mid SpdLim_{viol}, SafDst_{viol}, \mathbf{c}) \cdot P(Halted_{true} \mid$$
$$SpdLim_{viol}, SafDst_{viol}, \mathbf{c}) \cdot P(SafDst_{viol} \mid \mathbf{c}) +$$
$$P(TripDur_{true} \mid SpdLim_{viol}, SafDst_{ob}, \mathbf{c}) \cdot P(Halted_{true} \mid$$
$$SpdLim_{viol}, SafDst_{ob}, \mathbf{c}) \cdot P(SafDst_{ob} \mid \mathbf{c})$$

while the right member of the difference in Eq. 4.7 is

$$P(\mathbf{O}_{true}|N_{ob}, \mathbf{c}) = P(TripDur_{true}, Halted_{true}|SpdLim_{ob}, \mathbf{c}) =$$
$$P(TripDur_{true} \mid SpdLim_{ob}, SafDst_{viol}, \mathbf{c}) \cdot P(Halted_{true} \mid$$
$$SpdLim_{ob}, SafDst_{viol}, \mathbf{c}) \cdot P(SafDst_{viol} \mid \mathbf{c}) +$$
$$P(TripDur_{true} \mid SpdLim_{ob}, SafDst_{ob}, \mathbf{c}) \cdot P(Halted_{true} \mid$$
$$SpdLim_{ob}, SafDst_{ob}, \mathbf{c}) \cdot P(SafDst_{ob} \mid \mathbf{c})$$

Therefore the derivative of Eq. 4.7 for a norm node *SpdLim* in the Bayesian Network of Fig. 4.3 can be computed as:

$$P(TripDur_{true} \mid SpdLim_{viol}, SafDst_{viol}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{viol}, SafDst_{viol}, \mathbf{c}) \cdot P(SafDst_{viol} \mid \mathbf{c}) + P(TripDur_{true} \mid SpdLim_{viol}, SafDst_{ob}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{viol}, SafDst_{ob}, \mathbf{c}) \cdot P(SafDst_{ob} \mid \mathbf{c}) - P(TripDur_{true} \mid SpdLim_{ob}, SafDst_{viol}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{ob}, SafDst_{viol}, \mathbf{c}) \cdot P(SafDst_{viol} \mid \mathbf{c}) - P(TripDur_{true} \mid SpdLim_{ob}, SafDst_{ob}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{ob}, SafDst_{ob}, \mathbf{c}) \cdot P(SafDst_{ob} \mid \mathbf{c})$$

The *RRS* for a norm N determines the change in $P(N_{viol}|\mathbf{c})$ that is estimated, based on observed data from MAS execution, to be required in order to make the norm set $\mathcal{N}$ effective.

Naive sensitivity analysis suggests to change (increase or decrease) the violations of norms of the amount determined by the corresponding *RRS*s. The sign of the required revision strength determines whether it is required to reduce (negative *RRS*) or to increase (positive *RRS*) violations of a norm, i.e., it determines the direction of the required revision. The value of the *RRS* determines the intensity of the required change. For instance if $\Delta\theta_{N1_{viol|\mathbf{c}}} = +0.2$ and $\Delta\theta_{N2_{viol|\mathbf{c}}} = -0.5$, then the suggestion is to increase $P(N1_{viol})$ of 0.2 and to decrease $P(N2_{viol})$ of 0.5.

This strategy computes the *RRS* for a norm, without considering that a change could be applied, at the same time, also to other norms. In other words, the *RRS* for a norm N is computed as if no change in the probability of violating any other norm could happen (from this the term *naive*). However, when determining the *RRS* for a norm, *Naive sensitiity analysis* considers all possible values of the other norms. Therefore, this strategy may result robust to unexpected changes in the probability of violating other norms when changing the sanctions.

n-CPT sensitivity analysis. Determine the required revision strength for all norms together, by solving, following Chan et al. [76], equation 4.8 for the n parameters $\Delta\theta_{N1_{viol}}, ..., \Delta\theta_{Nn_{viol}}$. Let $co(\mathbf{N}, i)$ be the set of all possible combinations of i norm nodes from the set $\mathbf{N}$, and, given a set $\mathbf{M} = \{N1, ..., Nm\} \subseteq \mathbf{N}$ of norm nodes, let $\frac{\delta^i}{\delta\theta\mathbf{M}_{viol|\mathbf{c}}}$ be the Leibniz's notation for the i-th partial derivative $\frac{\delta^i}{\delta\theta N1_{viol|\mathbf{c}} \ldots \delta\theta Nm_{viol|\mathbf{c}}}$ for $N_j \in \mathbf{M}$.

$$P(\mathbf{O}_{true}|\mathbf{c}) + \sum_{i=1}^{n} \left[\sum_{\mathbf{M} \in co(\mathbf{N},i)} \left(\frac{\delta^i P(\mathbf{O}_{true}|\mathbf{c})}{\delta\theta\mathbf{M}_{viol|\mathbf{c}}} \cdot \prod_{N \in \mathbf{M}} \Delta\theta_{N_{viol|\mathbf{c}}} \right) \right] \geqslant t_{oa} \tag{4.8}$$

Solving equation 4.8 means to determine a list of n values $\Delta\theta_{N_{viol}}$, one for each norm node $N \in \mathbf{N}$. To do so, first of all it is required to compute: the n first partial derivatives $\frac{\delta P(\mathbf{O}_{true}|\mathbf{c})}{\delta\theta_{N_{viol|\mathbf{c}}}}$ (one for each norm $N \in \mathbf{N}$); the second partial derivatives for the $\binom{n}{2}$ possible combinations of two norm nodes from $\mathbf{N}$; the third partial derivatives for the $\binom{n}{3}$ possible combinations of three norm nodes from $\mathbf{N}$; and so on until the n-th partial derivative $\frac{\delta^n P(\mathbf{O}_{true}|\mathbf{c})}{\delta\theta_{N1_{viol|\mathbf{c}}} \ldots \delta\theta_{Nn_{viol|\mathbf{c}}}}$.

For instance, in the case of $\mathbf{N} = \{N1, N2\}$, we have that $n = 2$, $co(\mathbf{N}, 1) = \{\{N1\}, \{N2\}\}$, and $co(\mathbf{N}, 2) = \{\{N1, N2\}\}$, and inequality 4.8 corresponds to inequality 4.9.

$$\begin{aligned} & P(\mathbf{O}_{true}|\mathbf{c}) + \\ \frac{\delta P(\mathbf{O}_{true}|\mathbf{c})}{\delta \theta_{N1_{viol|\mathbf{c}}}} \cdot \Delta\theta_{N1_{viol|\mathbf{c}}} + & \frac{\delta P(\mathbf{O}_{true}|\mathbf{c})}{\delta \theta_{N2_{viol|\mathbf{c}}}} \cdot \Delta\theta_{N2_{viol|\mathbf{c}}} + \\ & \frac{\delta^2 P(\mathbf{O}_{true}|\mathbf{c})}{\delta \theta_{N1_{viol|\mathbf{c}}} \delta \theta_{N2_{viol|\mathbf{c}}}} \cdot \Delta\theta_{N1_{viol|\mathbf{c}}} \Delta\theta_{N2_{viol|\mathbf{c}}} \geqslant \quad t_{oa} \end{aligned} \tag{4.9}$$

The first partial derivatives in Eq. 4.8 can be computed as per Eq. 4.7, while the second partial derivative, in the case of two norms (as it is in Eq. 4.9), can be computed as per Eq 4.10.

$$\begin{aligned} & \frac{\delta^2 P(\mathbf{O}_{true}|\mathbf{c})}{\delta \theta_{N1_{viol|\mathbf{c}}} \delta \theta_{N2_{viol|\mathbf{c}}}} = \\ & P(\mathbf{O}_{true}|N1_{viol}, N2_{viol}, \mathbf{c}) + P(\mathbf{O}_{true}|N1_{ob}, N2_{ob}, \mathbf{c}) - \\ & P(\mathbf{O}_{true}|N1_{viol}, N2_{ob}, \mathbf{c}) - P(\mathbf{O}_{true}|N1_{ob}, N2_{viol}, \mathbf{c}) \end{aligned} \tag{4.10}$$

If we consider the running example from Fig. 4.3, the derivative in Eq. 4.10 can be computed as follows.

$$\begin{aligned} & P(TripDur_{true} \mid SpdLim_{viol}, SafDst_{viol}, \mathbf{c}) \cdot P(Halted_{true} \mid \\ & SpdLim_{viol}, SafDst_{viol}, \mathbf{c}) + \\ & P(TripDur_{true} \mid SpdLim_{ob}, SafDst_{ob}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{ob}, SafDst_{ob}, \mathbf{c}) - \\ & P(TripDur_{true} \mid SpdLim_{viol}, SafDst_{ob}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{viol}, SafDst_{ob}, \mathbf{c}) - \\ & P(TripDur_{true} \mid SpdLim_{ob}, SafDst_{viol}, \mathbf{c}) \cdot P(Halted_{true} \mid SpdLim_{ob}, SafDst_{viol}, \mathbf{c}). \end{aligned}$$

After determining the values of the opportune derivatives, as above reported, inequality 4.8 can be solved by solving the following optimization problem.

$$\begin{aligned} & \underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize}} \ f(\mathbf{x}) \\ & \text{subject to: } t_{oa} - f(\mathbf{x}) \leqslant 0 \end{aligned} \tag{4.11}$$

where $\mathbf{x} = (x_1, ..., x_n)$ is a vector of real values, such that x_i is a possible value for $\Delta\theta_{Ni_{viol|\mathbf{c}}}$ and $f(\mathbf{x})$ is the left member of inequality 4.8. Notice that the constraint to which the optimization problem is subject to corresponds to the canonical form of Eq. 4.8. Solving the optimization problem 4.11 means to determine the minimum values for the n parameters $\Delta\theta_{N1_{viol|\mathbf{c}}}, ..., \Delta\theta_{Nn_{viol|\mathbf{c}}}$ that satisfy the desired constraint of inequality 4.8 (i.e., the probability of achieving the objectives, after applying the required change in the probability of violating the enforced norms, is above the desired threshold t_{oa}).

Analogously to *naive sensitivity analysis*, *n-CPT sensitivity analysis* suggests to change (increase or decrease) the violations of norms of the amount of the corresponding *RRS*s determined by solving inequality 4.8. For instance, in the case of two norms, if $\Delta\theta_{N1_{viol|\mathbf{c}}} = +0.2$ and $\Delta\theta_{N2_{viol|\mathbf{c}}} = -0.5$, then the suggestion is to increase

$P(N1_{viol})$ of 0.2 and to decrease $P(N2_{viol})$ of 0.5. Differently from the previous strategy, however, such values are obtained taking into account the change applied at the same time to the probability of violating all norms (instead of applying a change only one norm at a time).

Category-based strategies

Category-based strategies classify norms into different categories, based on their exhibited norm violation and on their relationship with the system-level objectives discovered at run-time, and determine an adequate revision for each norm based on their category. We present two strategies of this type (*Synergy+MLE* and *State-based*), based on the two heuristic strategies presented in Chapter 3, and used to suggest a revision of requirements. In this chapter, we adapt them to support the revision of sanctions.

Synergy+MLE. This strategy is based on the pureBN strategy presented in Chapter 3. We distinguish between norms that are *more useful when obeyed* (*useful-ob* for brevity) or *more useful when violated* (*useful-viol*). Furthermore, norms can also be either *most likely obeyed* when the objectives are not achieved (*likely-ob* for brevity) or *most likely violated* (*likely-viol).* In order to distinguish between *useful-ob* and *useful-viol* we calculate the combined synergy $\mathbf{n}_d$ (as per Eq. 4.5). Norms with value *ob* in $\mathbf{n}_d$ are *useful-ob*, norms with value *viol* in $\mathbf{n}_d$ are *useful-viol.* In order to distinguish between *likely-ob* and *likely-viol*, instead, we determine the *most likely explanation* [186] **mle** for $\mathbf{O}_{false}$ in context $\mathbf{c}$, as follows (with **ov** defined as per Eq. 4.5).

$$\mathbf{mle} = argmax_{\mathbf{n} \in \mathbf{ov}}\ P(\mathbf{n}\ |\mathbf{O}_{false} \wedge \mathbf{c}) \tag{4.12}$$

Norms with value *ob* in **mle** are *likely-ob*, norms with value *viol* in **mle** are *likely-viol.*

Synergy+MLE suggests to increase violations of norms belonging to category *useful-viol* (*more useful when violated*); to reduce violations of norms belonging to both categories *useful-ob* and *likely-viol* (norms that are *more useful when obeyed*, but most likely *violated* when the objectives are not achieved); and to do nothing for, or reduce violations of, norms belonging to both categories *useful-ob* and *likely-ob* (norms that are *more useful when obeyed*, and most likely *obeyed* when the objectives are not achieved).

The original pureBN strategy [117] included the concept of *harmful norm*: a norm that is better when disabled. The suggestion of pureBN for harmful norms is to disable them. In this chapter, we only consider active norms and we focus on the sanction revision, thereby omitting specific suggestions for harmful norms. However, a suggestion of increasing violation of a norm N, may lead to enforce N with a sanction equals to 0. In this chapter, enforcing a norm N with a sanction of 0 corresponds to disabling N.

Finally, note that *Synergy+MLE* is a refinement of *Combined synergy* strategy. In addition to the combined synergy, this strategy also takes into account the most likely explanation for the objectives being not achieved, in terms of obedience or violation of norms.

State-based. This strategy, based on the STATEBASED strategy presented in Chapter 3, considers, in addition to the classification of norms described for strategy *Synergy+MLE*, information about the system state in context $\mathbf{c}$. In particular, as illustrated in Fig. 4.5, the system can be in four states with respect to the average norm obedience, calculated as the mean $ns = mean_{N \in \mathbf{N}} P(N_{ob}|\mathbf{c})$, and the objectives achievement probability $oa = P(\mathbf{O}_{true} \mid \mathbf{c})$.

- In state A, norms are sufficiently obeyed, but this does not lead to sufficient objectives achievement (i.e., $ns \geqslant t_{ns}$ and $oa < t_{oa}$ for some given t_{ns} and t_{oa}).
- In state B, norms are not sufficiently obeyed and also objectives are not achieved (i.e., $ns < t_{ns}$ and $oa < t_{oa}$).
- In state C, the objectives are achieved even though the norms are not obeyed (i.e., $ns < t_{ns}$ and $oa \geqslant t_{oa}$).
- In state D, (the desired state of the system) the norms are satisfied and the objectives are achieved (i.e., $ns \geqslant t_{ns}$ and $oa \geqslant t_{oa}$).

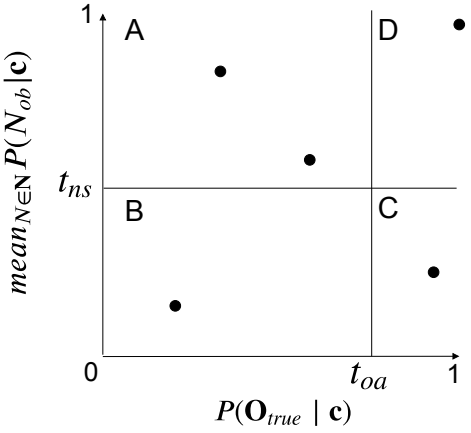


Figure 4.5: *System states (points) in four states (A-D) w.r.t. average norm obedience and objectives achievement probability.*

If the system is in state A, *State-based* suggests to increase violations of norms belonging to both categories *useful-viol* and *likely-ob*, i.e., norms that are more useful when violated but most likely obeyed when the objectives are not achieved, if any. Otherwise, *State-based* suggests to do nothing for (or to reduce violations of) the current norm set. In this case, there is probably some aspect of the system that has not been considered during its design, for the current norms are mostly obeyed and they are most useful when obeyed, but the system-level objectives are not achieved as desired. If the system is in state B, *State-based* suggests to reduce violations of norms belonging to both categories *useful-ob* and *likely-viol*, i.e., norms that are more useful when obeyed but most likely violated when the objectives are not achieved. It also

suggests to increase violations of norms belonging to category *useful-viol*, i.e., norms that are more useful when violated. If the system is in state C, finally, *State-based* suggests to increase violations of norms belonging to both categories *useful-viol* and *likely-viol*, if any. Otherwise it suggests to decrease violations of norms belonging to both categories *useful-ob* and *likely-viol*.

While *Synergy+MLE* suggests for all the norms in $\mathcal{N}$ the most adequate revision to perform, *State-based* considers the global state of the system and suggests to revise only a specific category of norms at every iteration (for the norms that do not belong to the category mentioned above it is suggested to do nothing). In case of high number of norms enforced, this strategy may significantly reduce the number of revisions that need to be performed at every step.

4.5.2 Sanctions Revision

Consider a norm $N = (p, s)$ and a revision of it $N' = (p, s')$, with $s' \neq s$. Let $P(N_{viol}|\mathbf{c})$ be N's exhibited norm violation (i.e., the probability of violating N monitored during system's execution) in context $\mathbf{c}$. We call *applied revision strength* the difference $UB(N'_{viol}, \mathcal{N}) - P(N_{viol}|\mathbf{c})$ between the upper bound UB for violation of N' (as per Sec. 4.4.2) and the N's exhibited norm violation. For instance, in the example reported in Fig. 4.4, supposing that when enforcing $N_1 = (sp_{50}, 1)$ and $N_2 = (dist_2, 0)$ the exhibited norm violation of N_1 is 0.3, the applied revision strength when revising N_1 into $N'_1 = (sp_{50}, 2)$ is $0 - 0.3 = -0.3$, while the applied revision strength when revising N_1 into $N'_1 = (sp_{50}, 0)$ is $1 - 0.3 = 0.7$.

The strategies described in Sec. 4.5.1, provide, for each norm N in $\mathcal{N}$, a suggestion such as *reduce*/*increase* violations of N, do *nothing* with N, *reduce*/*increase* violations of N of a certain *amount RRS* (as per Sec. 4.5.1). Given these suggestions, and all possible sets of sanctions $\mathcal{S}$ that can be used to enforce norms in $\mathcal{N}$, we need to find a new set of sanctions such that the *applied revision strength* satisfies (as much as possible) the given suggestions.

A trivial solution is to systematically go through all elements in $\mathcal{S}$ until the desired sanction set (if it exists) is found. Such solution is however computationally expensive, as the number of possible sanction sets is $(max(\mathcal{B}) + 2)^n$, with $max(\mathcal{B}) + 2$ maximum budget among all agent types ($max(\mathcal{B})$) plus sanction 0 and sanction $max(\mathcal{B}) + 1$, and n number of norms.

In the following, we propose a simple alternative way to explore the search space $\mathcal{S}$ that can be used in case of a population of consistent agent types as per Def. 13. With a population of consistent agent types, according to Prop. 3, the upper bound of the probability of violating norms decreases monotonically when any sanction increases. This means that given a sanction set, and the exhibited norm violation for each enforced norm, if we desire to apply a negative revision strength, we need to move towards higher values of sanctions. To apply a positive revision strength, instead, we could change in any way the sanctions (even though typically we should move towards lower values of sanctions), since the currently exhibited norm violation could be lower than the upper bound of norm violation with an higher sanction.

Under the assumption of consistent agent types, we can reduce therefore the exploration of $\mathcal{S}$ by directing the search towards the desired values of sanctions. For

instance, suppose to have two norms $N1 = (p_1, s_1)$ and $N2 = (p_2, s_2)$, and a list of suggestions $sugg = (reduce, increase)$ for a context $\mathbf{c}$ (i.e., it is suggested to reduce violations of norm $N1$ and to increase violations of norm $N2$ in context $\mathbf{c}$). Given $sugg$, we need to look for a new sanction set $\{s'_1, s'_2\}$ such that $UB(N1'_{viol}, \mathcal{N}') < P(N1_{viol}|\mathbf{c})$ and $UB(N2'_{viol}, \mathcal{N}) > P(N2_{viol}|\mathbf{c})$, with $\mathcal{N}' = \langle N1', N2' \rangle$. We can therefore reduce the search space to the subset of $\mathcal{S}$ such that $s'_1 \geqslant s_1$ and $s'_2 \neq s_2$.

Algorithm 4 reports the pseudo-code of a procedure to perform such search.

Notice that if preferences are not consistent, we have no guarantees that by moving towards higher values of sanctions we will not increase violations of norms, since Prop. 3 does not hold in the general case (i.e., for preferences that are not consistent). Despite this, one may still heuristically explore $\mathcal{S}$ by using Algorithm 4, also when not all preferences are consistent.

Algorithm 4 is invoked when a suggestion of norm revision has been determined with one of the strategies of Sec. 4.5.1 after a norm revision is triggered, and there is at least one sanction set that has not been tried previously in context $\mathbf{c}$. If a sanction set has already been tried, we know it is not effective (otherwise no further norm revision would have been triggered). If all possible sanction sets have been already tried (omitted from Algorithm 4), then the sanction set that, when enforced, maximizes $P(\mathbf{O}_{true} \mid \mathbf{c})$ is selected.

The algorithm takes as input: the list of currently enforced sanctions cs; the exhibited violation of the enforced norms E; the list $sugg$ of suggestions obtained with one of the strategies of Sec. 4.5.1 (a value *reduce* (or *increase*, or *nothing*) in $sugg[i]$ corresponds to a suggestion to reduce (increase, or do nothing with) violations of the i-th norm); a matrix UB containing the upper bounds for norms violations as per Fig. 4.4; a list RRS of required revision strengths (empty if no sensitivity-based strategy is used); and the context $\mathbf{c}$. As output, Algorithm 4 returns a (possibly new) list of sanctions to use to enforce norms in context $\mathbf{c}$.

The algorithm explores the possible sanction sets starting from the current sanction set cs. The first step is to determine the subset of $\mathcal{S}$ to explore. Notice that, if a *reduce* suggestion has been given for norm i (i.e., $sugg[i] = reduce$), the new sanction for norm i must be greater of equal than the current one (i.e., $ns[i] \geqslant cs[i]$). This means that $ns[i]$ has to be equal to $cs[i] + ch$, with $ch \geqslant 0$ amount of change. Conversely, if $sugg[i] = increase$, $ns[i]$ has to be equal to either $cs[i]+ch$ or $cs[i]-ch$. If we put these cases together (line 6 of Algorithm 4), $ns[i]$ has to be equal to $cs[i]+o[i]\cdot ch$, with $o[i] \in \{0, 1\}$ if $sugg[i] = reduce$, and $o[i] \in \{0, -1, 1\}$ if $sugg[i] = increase$, and $ch > 0$ amount of change.

Variable *comb* (line 3) is a set of all possible combinations of operators $o[i]$ for each norm i, obtained from their suggestion $sugg[i]$ (see $op[sugg[i]]$, which retrieves from the labeled set *op* declared at line 2, the opportune list of operators given suggestion $sugg[i]$). For instance, supposing to have two norms $N1 = (p_1, s_1)$ and $N2 = (p_2, s_2)$, and a list of suggestions $sugg = (reduce, increase)$, we have that $op[sugg[1]] = [0, 1]$ and $op[sugg[2]] = [0, -1, 1]$ and *comb* is the set of all possible combinations of operators in $op[sugg[1]]$ and $op[sugg[21]]$, i.e., $\{(0,0), (0,-1), (0,1), (1,0), (1,-1), (1,1)\}$, such that given a certain element $o \in comb$, $o[i]$ is the operator to apply to the change of sanction $cs[i]$.

The algorithm iterates through all possible changes that can be applied to sanc-

tions (line 4). For each possible change, the algorithm iterates through all possible new sanction sets that can be obtained with the combinations of operators in *comb* (lines 5-6). Notice that, by iteratively increasing the change, we explore the search space at increasing distance from the current sanction set. This means that if the algorithm finds a new (function isNewSanctionSet at line 7) sanction set ns that satisfies the given suggestions (function suggSat at line 8), such sanction set is also the closest possible to the current one.

Finally, if no new sanction set satisfying the suggestions is found, the current sanction set is returned. In this case, in our framework a random sanction set never tried before is enforced in context **c**.

Notice also that function suggSat (line 8), whose purpose is to verify that a proposed sanction set satisfies the given suggestions, does not need to require that *all* suggestions are perfectly satisfied. In particular, especially when suggestions include also a required revision strength (i.e., when using sensitivity-based strategies), it may be more useful to search for a good-enough sanction set. For our experiments, described in Sec. 4.6, when list *RRS* is not empty, we keep track of the best new sanction set found so far (if not all suggestions are satisfied) and for every new sanction set tested we require at least 80% of suggestions to be satisfied. Furthermore in case of suggestion *nothing*, since unlikely, in our experiments, that the exhibited probability of a norm exactly corresponds to a value on its upper bound, we accept also a reduction of the probability of violating the norm of a small ϵ (we used $\epsilon = 0.1$).

Algorithm 4 The algorithm for the selection of a new sanction set

Input: *cs* the current sanction set,
E the norms' exhibited violation,
sugg the n suggestions,
UB the upper bound matrix
RRS a (possibly empty) list of required revision strengths
c the context
Output: a new sanction set *sugg* to enforce in **c**

```
1: function getSanctionSet(cs, E, sugg, UB, RRS, c)
2:     op ← {reduce : [0, 1], increase : [0, −1, 1], nothing : [0, 1]}
3:     comb ← ×_{i=1}^{n} op[sugg[i]]
4:     for all ch ∈ [1, max(B)] do
5:         for all o ∈ comb do
6:             ns ← [cs[i] + o[i] · ch | ∀i ∈ [1, n]]
7:             if isNewSanctionSet(ns,c) then
8:                 if suggSat(cs, ns, sugg, UB, E, RRS, c) then
9:                     return ns
       return cs
```

After enforcing the new norm set $\mathcal{N}'$, obtained by revising the sanctions of norms in $\mathcal{N}$ according to the new sanction set obtained from Algorithm 4, we monitor the new behavior of the agents and detect the new exhibited norm violation $P(N'_{viol}|\mathbf{c})$, for each norm $N' \in \mathcal{N}'$. We call *actual revision strength* the difference $P(N'_{viol}|\mathbf{c}) - P(N_{viol}|\mathbf{c})$

between the exhibited norm violation of N' and N, with $N' = (p, s')$and $N = (p, s)$.

4.6 Experimentation

We report on an experiment that investigates the process through which the norm-based supervision mechanism of Sec. 4.4 identifies an optimal system configuration. The object of our study consists of the strategies for norm revision proposed in Sec. 4.5. In particular, we study the process through which the norm-based supervision mechanism identifies an optimal system configuration when employing each of the six proposed strategies as possible *informed heuristics* for defining the neighborhood of a configuration, i.e., the configurations where the sanctions of the enforced norms are revised as suggested by the heuristics.

We compare the results in terms of *convergence speed*. The convergence speed measures the number of steps (i.e., revisions of the sanctions of norms triggered) required by the heuristic strategies to make the norms effective in achieving the system-level objectives. This allows us to study the time efficiency of the norm revision strategies in refining sub-optimal norms at run-time.

4.6.1 Experimental Setting

Our experiment is run through a simulation[5] of the ring road scenario described in Sec. 4.3. Our implementation of the norm-based supervision mechanism of Sec. 4.4, as a modified version of hill climbing, is called SASS (Supervisor of Autonomous Software Systems)[6]. The supervisor performs a local search and stops when either (i) all the system configurations have been tried; or (ii) a local optimum (system configuration) is found that has objectives achievement probability $oa = P(\mathbf{O}_{true})$ above the desired threshold t_{oa}. The objectives achievement probability of a certain system configuration is *not* known to SASS before the configuration is actually enforced. Such probability is determined at run-time from simulation data, given the chosen system configuration. In this experimental setting, the last system configuration that is selected before stopping is called *optimal*, since either the objectives achievement is above the desired threshold or there is no other better configuration.

In the ring road scenario, we consider the two contextual variables *Vehicle density*, which can be *low* (40 cars on the ring road) or *high* (80 cars)[7]; and *Obstacle*, which is *true* when an obstacle is placed on the outer lane of the ring road. Each car in the simulation is an agent that acts according to its specific characteristics, beliefs and preferences. At each simulation step, every agent also deliberates about a number of things, including its desired speed and the minimum safety distance, whether and how much to accelerate or decelerate, whether to change lane to surpass or to move back to the outer lane, whether to activate the turn signals. Agents' decisions are based on their own internals, which are specific for each agent and unknown to the norm revision mechanism. In our simulations, when an agent equally prefers two

[5]For our experiment, we used the SUMO traffic simulator [184] and CrowdNav+RTX [245]

[6]The source code and the material for experiments' replication can be found in [120].

[7]The density values have been determined empirically based on the size of the ringroad used for the experiments.

alternatives x and y concerning the speed and safety distance (i.e., $x \sim y$), the agent applies a deterministic choice to determine what state of affair to pursue (i.e., simply the first one in the representation of the alternatives), instead of random choice.

In order to define norms and agents' preferences, we consider the set of propositional atoms $L = \{sp_{15}, sp_8, sp_3, dist_{0.5}, dist_1, sp_2\}$, with $AL = (L_1, L_2)$ and $L_1 = \{sp_{15}, sp_8, sp_3, \}$ and $L_2 = \{dist_{0.5}, dist_1, dist_2\}$. Each element in L_1 represents a speed in m/s and each element in L_2 represents a safety distance in meters. Furthermore, we consider a language $\mathcal{B} = \{0, 1, 2\}$ for defining budgets and a language $\mathcal{S} = \{0, 1, 2, 3\}$ for defining sanctions of norms.

Agent Types

We experiment with four types of rational agents with consistent preferences (as per Def. 13). In the following we briefly describe such types, and we report in Appendix B the full preferences.

- *BraveRich* is a consistent basic preference that adheres to Def. 10b, i.e., where alternatives are ordered by propositional atom. It describes an agent type with a maximum budget of 4, that prefers to drive fast and to keep a short safety distance, and that gives priority to the short safety distance rather than to driving fast.

- *BraveMiddleClass* is a consistent preference composed by two basic preferences. The first basic preference $(A_1, \succeq_1)$ adheres to Def. 10b. The alternatives in A_1 are such that $A_1 = \{(\langle p_1, b_1\rangle, \ldots, \langle p_n, b_n\rangle) \mid p_i \in L_i \ \& \ (b_1, \ldots, b_n) \in BL_1\}$, with $BL_1 = \{(0,0), (0,1), (1,0), (1,1)\}$. The second basic preference $(A_2, \succeq_2)$ adheres to Def. 10a. The alternatives in A_2 are such that $A_2 = \{(\langle p_1, b_1\rangle, \ldots, \langle p_n, b_n\rangle) \mid p_i \in L_i \ \& \ (b_1, \ldots, b_n) \in BL_2\}$, with $BL_2 = \{(0,2), (1,2), (2,2), (2,1), (2,0)\}$. *BraveMiddleClass* describes an agent type similar to *BraveRich*, but that is willing to pay no more than 2 for a certain state of affairs. The alternatives in A_2 are ordered by required budget and, for consistency, they maintain the same relative order as in A_1.

- *BravePoor* is a consistent basic preference ordered by required budget, as per Def. 10a. It describes an agent type that equally prefers to drive fast or slow and to keep a short or long safety distance, but is not willing to pay anything to reach any state of affairs.

- *Cautious* is a consistent basic preference ordered by required budget, as per Def. 10a. It describes an agent type that equally prefers to drive slow or fast and to keep a long or short safety distance, and is not willing to pay anything to reach any state of affairs. Notice that this preference is equivalent to *BravePoor*, however due to the deterministic mechanism of choice of an alternative that our agents employ (i.e., the first one in the representation of the alternatives), these two agent types will exhibit different behaviors at run-time. For instance, even though states of affairs where sp_{15} and $dist_{0.5}$ hold are equally preferred to state of affairs where sp_3 and $dist_{0.5}$ hold, in both preferences, and they could both be chosen in the case of random choice, in our simulation, given enough

budget, *BravePoor* will aim at a state of affair where sp_{15} and $dist_{0.5}$ hold, while *Cautious* will aim at a state of affair where sp_3 and $dist_{0.5}$ hold.

We consider three distributions of types of agents:

- *uniform*, the entire population of agents is uniformly distributed across the four types described above.
- *mostly compliant*, 75% of agents belongs to type *Cautious* and the rest is uniformly distributed across the remaining types.
- *mostly violating*, 75% of agents belongs to type *BraveRich* and the rest is uniformly distributed across the remaining types.

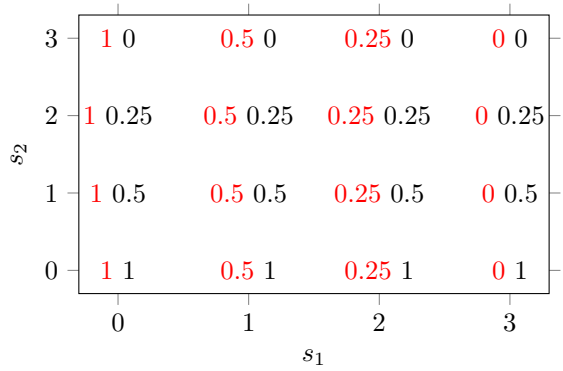
Note that despite our estimation of the preferences of the agents concerning speed and safety distance, we do not have any control on the exact speed or safety distance of the agents, which is internally and opaquely set by the agents, together with the rest of their behaviors.

Norms

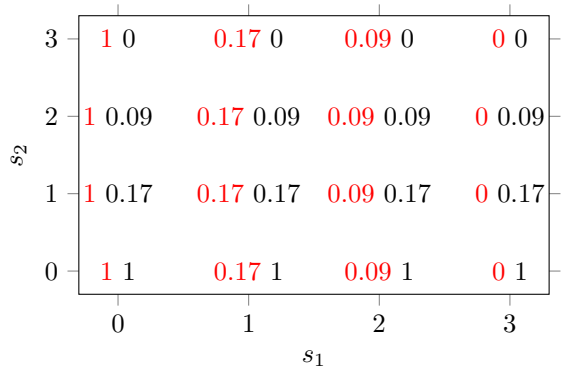
We consider the following four ordered norm sets: $\mathcal{N}_{31} = \langle SpdLim_3, SafDst_1 \rangle$, $\mathcal{N}_{32} = \langle SpdLim_3, SafDst_2 \rangle$, $\mathcal{N}_{81} = \langle SpdLim_8, SafDst_1 \rangle$, $\mathcal{N}_{82} = \langle SpdLim_8, SafDst_2 \rangle$, with $SpdLim_x = (sp_x, s_1)$, $SafDst_y = (dist_y, s_2)$, $x \in \{3, 8\}$, $y \in \{1, 2\}$ and s_1 and s_2 sanctions in $\mathcal{S}$.

Fig. 4.6 illustrates the upper bounds of the probability of violating the two norms $SpdLim_x$ and $SafDst_y$ defined above (as per Sec. 4.4.2) for the three agent type distributions. Notice that the reported upper bounds hold for all combinations of the values x and y above defined (i.e., values of speed limit and minimum safety distance). This is due to the types of agents that we considered for our experiments. *BraveRich* prefers to keep a speed of 15 m/s by maintaining a short safety distance and it is willing to pay a sanction of 2 for each of these aspects. When the sanction of a norm is above 2 this agent is compliant with the norm, *regardless of the value of the speed limit*, because the agent has no budget for violating the norm. *BraveMiddleClass* is analogous to *BraveRich* but with a maximum budget of 1 for the violation of a norm: up to sanction 1 *BraveMiddleClass* has reason to violate a norm (it also prefers to go at a speed of 15 m/s by maintaining a short safety distance), while when the sanction of a norm is above 1 *BraveMiddleClass* is compliant. Finally *BravePoor* and *Cautious* have reason to violate the norms only when their sanctions is 0. With higher sanctions, these agent types are compliant.

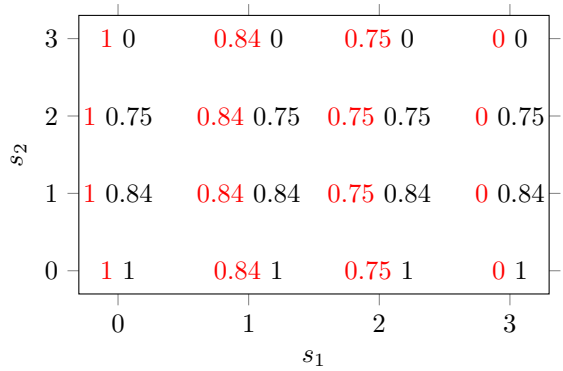
Furthermore notice that, since all the agent types that we considered are consistent as per Def. 13, the upper bounds reported in Fig. 4.6 satisfy Prop. 3: when increasing the sanction of only one norm the upper bound of violating the other norm never increases. This allows us to take advantage, in our experiments, of Algorithm 4 for the selection of a new sanction set.



(a) *uniform*



(b) *mostly compliant*



(c) *mostly violating*

Figure 4.6: *Upper bound of the probability of violating norms $SpdLim_x = (sp_x, s_1)$ (red) and $SafDst_y = (dist_y, s_2)$ (black) with different agent type distributions. In each subfigure the x-axis represents the sanction s_1 of norm $SpdLim_x$, while the y-axis represents the sanction s_2 of norm $SafDst_y$.*

4.6.2 Experiments

By combining the three distributions of agents of Sec. 4.6.1 with the four norm sets of Sec. 4.6.1, we derived 12 different experiments. We ran a simulation of the ring road for each of the 12 experiments and we collected data about norm obedience and objective achievement in the four different operating contexts $\mathbf{c1} = \mathit{VehicleDensity}_{low} \wedge \mathit{Obstacle}_{false}$, $\mathbf{c2} = \mathit{VehicleDensity}_{low} \wedge \mathit{Obstacle}_{true}$, $\mathbf{c3} = \mathit{VehicleDensity}_{high} \wedge \mathit{Obstacle}_{false}$, $\mathbf{c4} = \mathit{VehicleDensity}_{high} \wedge \mathit{Obstacle}_{true}$. This means that during a simulation, the contexts in which the cars on the ring road operate changes three times (for a total of fours different operating contexts in each simulation). During the simulations, we monitored the behavior of the cars and sanctioned each car that violated one of the enforced norms. A car sanctioned for the violation of a norm N was not sanctioned anymore for violations of norm N until it completed a full loop of the ring road. The Boolean value of the system-level objectives was measured every 25 simulation steps. The objective *TripDur* was considered achieved if, on average in the 25 steps, the cars on the ring road took less than 2.5 times the theoretical average trip time[8] to complete a loop of the ring road. The objective *Halted* was considered achieved if, on average in the 25 steps, less than $x\%$ of cars were halted on the ring road, with $x = 25$ if the density of vehicles on the ring road is high, and $x = 5$ if the density of vehicles is low[9]. A car in SUMO is considered halted if its speed is below $0.1m/s$. Cars could be halted on the ring road for several reasons. For example, the presence of an obstacle may force them to stop and wait for the right moment to surpass the obstacle or traffic waves may force cars to temporary slow down significantly to avoid collisions.

In every experiment that we perform, the system has n^m possible configurations, with n possible sanction sets and m different operating contexts. Since the speed of convergence to an optimal solution depends on the initial system configuration (i.e., a different amount of revisions may be required starting from different initial configurations), we execute each strategy starting from each possible configuration and we calculate statistics information (i.e., median, maximum, mean and standard deviation) concerning the convergence speed in the different executions. To keep our experimentation's time manageable, in our experiments we considered only 2 of the 4 operating contexts: **c2** and **c3**. This allowed us to reduce the number of possible configurations from 16^4 to $16^2 = 256$: 16 possible sanction sets for the enforced norms in any of the 2 contexts.

Fig. 4.7 shows the probability $P(\mathbf{O}_{true})$ obtained with the 256 configurations in each of the 12 experiments and highlights the optimal configurations (the configurations s.t. $P(\mathbf{O}_{true}) \geqslant t_{oa}$). Every dot in Fig. 4.7 represents the probability of achieving the objectives during a simulation with a certain system configuration (i.e., $P(\mathbf{O}_{true})$), considering both the contexts **c2** and **c3**. In each sub-figure (one per experiment) we see therefore 256 dots, one per system configuration. Notice that in the 12 experiments, the distribution of the 256 configurations w.r.t. the probability

[8]The theoretical trip time is $\sum_{t_i \in \mathcal{T}} d_i \times t_{i,\mathcal{N}}$, with $\mathcal{T}$ being the set of agent types, d_i being the percentage of agents of type t_i, and $t_{i,\mathcal{N}}$ being the theoretical time needed by t_i to complete a loop in case of free ring road when norm set $\mathcal{N}$ is enforced.

[9]Values for the evaluation of the objectives were determined based on some preliminary experimentation with the ring road simulation in order to retrieve a variegate set of experiments.

of achieving the system-level objectives is different. In other words, a certain system configuration c (i.e., enforcing norms with certain sanctions in the two contexts **c2** and **c3**) can be effective in an experiment but ineffective in another experiment. This makes the 12 experiments independent, thereby increasing the generality of our results.

For each of the 12 experiments, we defined a different t_{oa} as indicated in Table 4.2, which summarizes the entire experimental setting. The different thresholds allow us to test our strategies with different degrees of difficulty (i.e., number of optimal configurations to be found).

Exp.	ID	Agents distribution	Norm set	Contexts	t_{oa}	Optimal
1	*DUN31*	*uniform*	$\mathcal{N}_{31}$	**c2**, **c3**	0.99	84/256 (32.8%)
2	*DUN32*	*uniform*	$\mathcal{N}_{32}$	**c2**, **c3**	0.99	48/256 (18.7%)
3	*DUN81*	*uniform*	$\mathcal{N}_{81}$	**c2**, **c3**	0.9	6/256 (2.3%)
4	*DUN82*	*uniform*	$\mathcal{N}_{82}$	**c2**, **c3**	0.7	8/256 (3.1%)
5	*DCN31*	*mostly compliant*	$\mathcal{N}_{31}$	**c2**, **c3**	0.99	72/256 (28.1%)
6	*DCN32*	*mostly compliant*	$\mathcal{N}_{32}$	**c2**, **c3**	0.99	60/256 (23.4%)
7	*DCN81*	*mostly compliant*	$\mathcal{N}_{81}$	**c2**, **c3**	0.79	4/256 (1.6%)
8	*DCN82*	*mostly compliant*	$\mathcal{N}_{82}$	**c2**, **c3**	0.5	17/256 (6.6%)
9	*DVN31*	*mostly violating*	$\mathcal{N}_{31}$	**c2**, **c3**	0.99	24/256 (9.4%)
10	*DVN32*	*mostly violating*	$\mathcal{N}_{32}$	**c2**, **c3**	0.9	96/256 (37.5%)
11	*DVN81*	*mostly violating*	$\mathcal{N}_{81}$	**c2**, **c3**	0.6	18/256 (7%)
12	*DVN82*	*mostly violating*	$\mathcal{N}_{82}$	**c2**, **c3**	0.8	3/256 (1.2%)

Table 4.2: *The setting of the 12 experiments.*

4.6.3 Analysis of the Results

Table 4.3 reports the results concerning the steps required by the supervision mechanism to find an optimal configuration in the 12 experiments when employing each of the six proposed revision strategies. In particular, we report the median, the maximum, the average, and the standard deviation of the number of steps. We highlight the best performing strategies in each experiment.

On average, all the strategies required a limited number of steps to find an optimal configuration in almost all experiments. In the 12 experiments, while the number of optimal configurations to be found ranges from 3 to 96 out of 256 configurations, on average the strategies never required more than 52 steps to find one of those configurations (see columns Avg (σ) in Table 4.3, where σ is the standard deviation), with a minimum of 0 for all strategies (trivially in the cases the initial configuration is optimal, not reported in Table 4.3), a maximum of 218 in the most difficult scenario (see columns Max of experiment *DVN82*), and a median value never above 35 steps.

If we look at the average values, the strategy that performed less well in the 12 experiments is *Naive sensitivity analysis*, which, in order to find an optimal configuration among the 256 possible configurations, required an average number of steps between 1 and 52. The strategy that, on average, performed best, instead, is *n-CPT sensitivity analysis*, requiring an average number of steps between 2 and 12. In particular, these

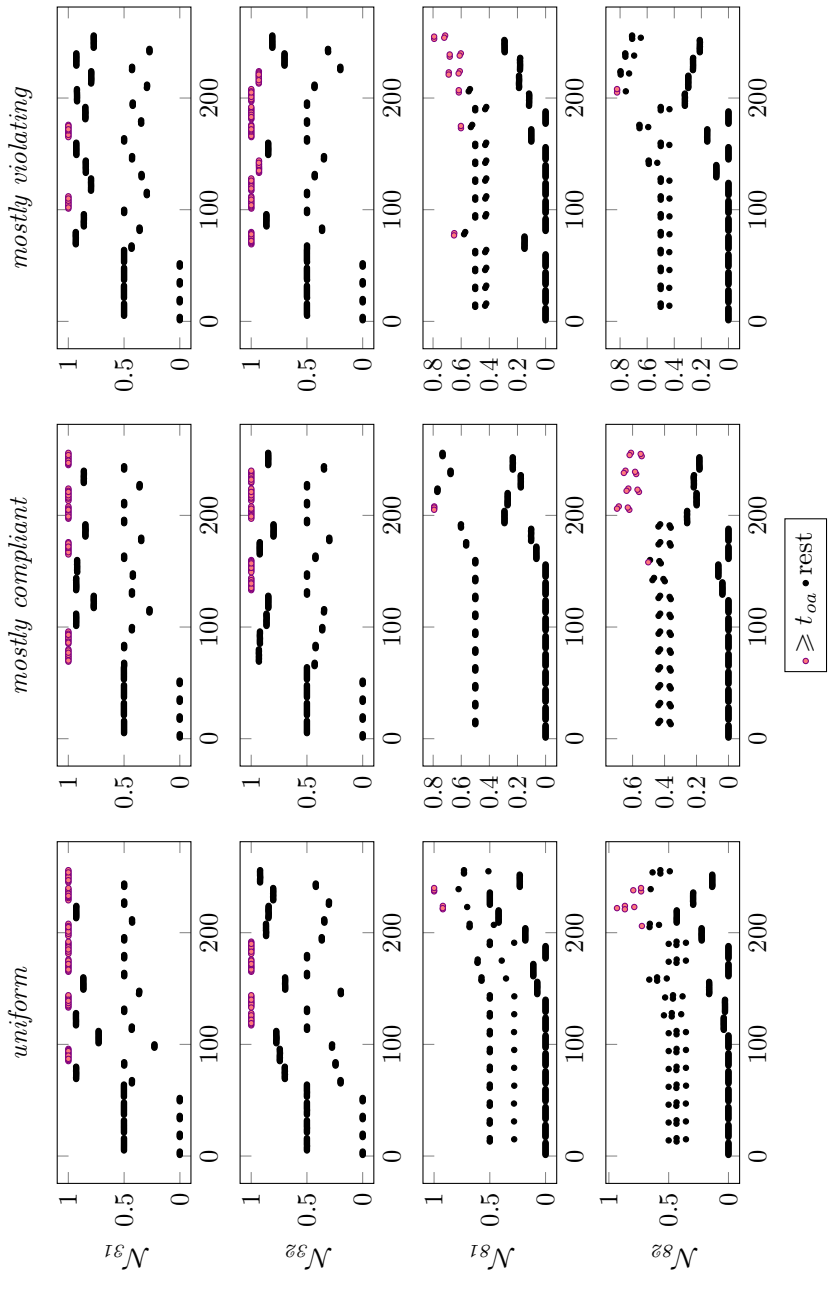


Figure 4.7: *Probability of objectives achievement (y-axis) for the 256 tried configurations (x-axis) in the 12 experiments.*

Table 4.3: *Comparison of the strategies in terms of number of steps required to find an optimal solution.*

	Naive synergy			*Combined synergy*			*Naive sentitivity*			*n-CPT sensitivity*			*Synergy+MLE*			*State-based*		
Exp.	Median	Max	Avg (σ)	Median	Max	Avg (σ)	Median	Max	Avg (σ)	Median	Max	Avg (σ)	Median	Max	Avg (σ)	Median	Max	Avg (σ)
DUN31	1.00	4	1.13 (1.03)	1.00	4	1.13 (1.03)	1.00	6	1.43 (1.40)	2.00	8	2.20 (1.98)	1.00	4	1.13 (1.03)	1.00	4	1.13 (1.03)
DUN32	3.00	14	3.15 (2.79)	3.00	14	3.12 (2.68)	2.00	11	2.48 (2.34)	4.00	8	3.40 (2.26)	2.00	13	2.79 (2.57)	2.00	11	3.12 (2.82)
DUN81	5.00	59	5.51 (5.63)	7.00	93	9.12 (9.46)	15.00	160	16.50 (15.45)	14.00	19	12.41 (4.21)	6.00	98	7.08 (7.50)	5.00	152	5.81 (10.10)
DUN82	6.00	23	6.05 (4.15)	6.00	61	6.91 (5.91)	7.00	42	6.54 (4.48)	13.00	19	11.72 (4.44)	7.00	22	6.75 (4.13)	6.00	25	6.43 (3.94)
DCN31	1.00	4	1.19 (1.00)	1.00	4	1.19 (1.00)	1.00	4	1.38 (1.14)	2.00	6	1.56 (1.34)	1.00	4	1.19 (1.00)	1.00	4	1.25 (1.03)
DCN32	2.00	11	2.10 (2.06)	2.00	9	2.09 (1.96)	3.00	10	3.29 (2.69)	4.00	10	3.82 (2.76)	2.00	8	2.11 (1.88)	1.00	9	1.86 (1.82)
DCN81	22.00	216	33.44 (36.92)	21.00	149	28.00 (24.79)	24.00	198	36.64 (33.58)	9.00	13	7.97 (2.64)	22.00	178	32.49 (30.12)	8.50	104	10.44 (10.36)
DCN82	3.00	8	3.39 (2.19)	7.00	14	5.95 (3.78)	9.00	12	6.68 (4.31)	8.00	12	6.79 (2.99)	7.00	15	6.26 (3.82)	5.00	11	4.57 (2.84)
DVN31	7.00	18	6.89 (4.72)	7.00	20	7.14 (4.90)	5.00	22	5.70 (4.27)	6.00	10	4.62 (2.94)	7.00	19	7.23 (4.95)	7.00	20	7.01 (4.66)
DVN32	1.00	7	1.44 (1.55)	1.00	6	1.43 (1.52)	1.00	8	1.38 (1.62)	1.00	6	1.36 (1.37)	1.00	8	1.23 (1.44)	1.00	5	1.09 (1.14)
DVN81	4.00	12	5.03 (3.56)	7.00	16	6.70 (4.74)	3.00	6	3.11 (1.71)	6.00	13	5.78 (2.74)	7.00	16	6.61 (4.63)	7.00	15	6.71 (4.70)
DVN82	8.00	218	20.63 (33.75)	20.00	204	38.43 (44.11)	35.50	159	52.32 (42.90)	9.00	162	12.20 (18.37)	19.00	217	35.97 (41.52)	19.50	191	32.75 (38.35)

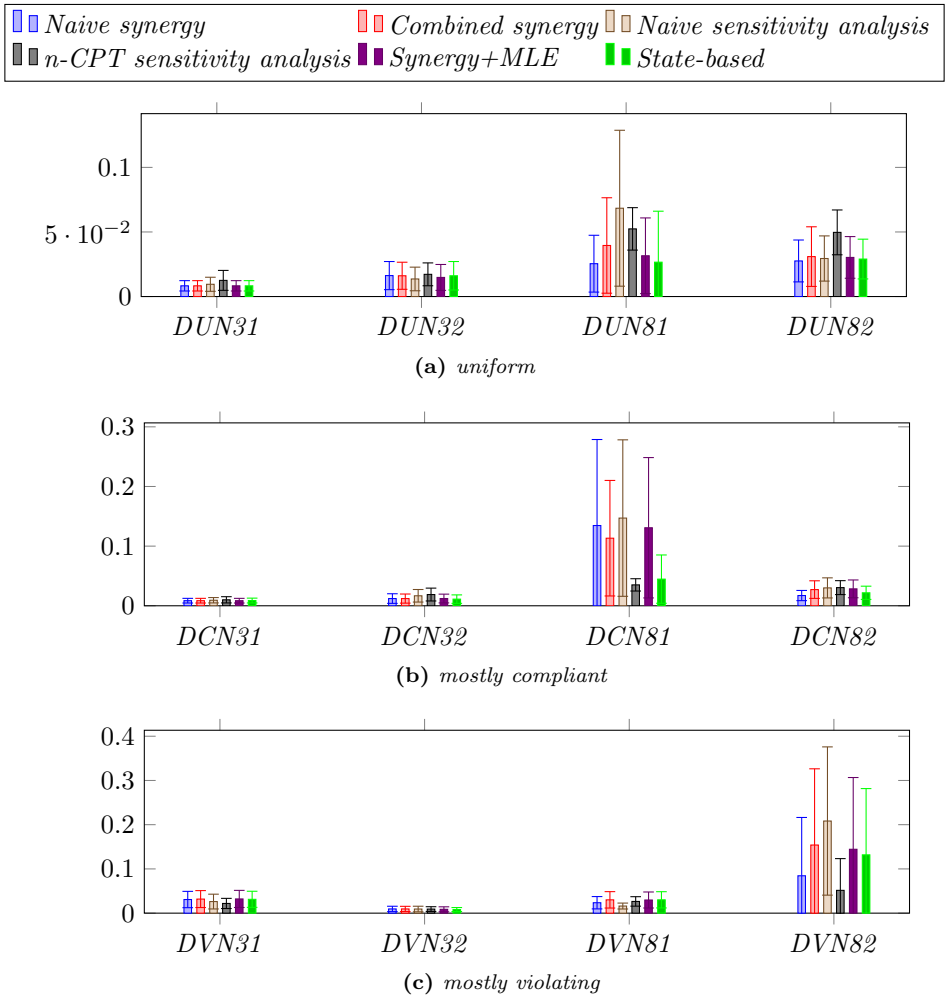
results show that when using *n-CPT sensitivity analysis*, on average, about 6 norm revisions were triggered by the norm-based supervision mechanism before finding a configuration where the system-level objectives were achieved as desired.

Despite *n-CPT sensitivity analysis* performed, on average, better than the other strategies in the 12 experiments, the results show that using that strategy was mostly advantageous when very few configurations were optimal among all the possible ones. In particular, *n-CPT sensitivity analysis* appeared to be more effective than the other strategies when the number of optimal configurations was lower than 2% of all the configurations. For instance, in experiment *DCN81* (1.6% of configuration are optimal), never more than 13 steps were required to find an optimal configuration when employing *n-CPT sensitivity analysis*, while the other strategies required a maximum number of steps between 104 and 216. Furthermore, while the median number of step is 9 with *n-CPT sensitivity analysis*, the median number of steps with the other strategies is more than twice. One exception is *State-based*, which in such experiment required an average number of steps similar to *n-CPT sensitivity analysis* and an even lower median. *State-based*, however, exhibited an higher variance, requiring in some executions up to 104 steps. In experiment *DVN82* (1.2% of configuration are optimal), while all other strategies (including *State-based*) required an average number of steps between 20 and 52, *n-CPT sensitivity analysis* was able to find on average an optimal configuration in about 12 steps.

If we consider, instead, simpler experiments (e.g., *DUN31* or *DCN31*), *n-CPT sensitivity analysis* did not outperform significantly the other strategies. In fact, if we consider the average number of steps, among all the strategies, *Naive synergy* outperformed (even though by few steps) all the others in 5 experiments, requiring in all of them less then 6 steps to find an optimal configuration. Furthermore in 8 experiments the average number of steps required by *Naive synergy* was below the average between the different algorithms. *State-based* had similar performances to *Naive synergy* and, even though it was the absolute best strategy in only 3 experiments in terms of average number of steps, in 8 experiments out of 12 it exhibited the lowest median value.

Fig. 4.8 plots the percentage of configurations explored in the 12 experiments by the six strategies before finding an optimal one. In most experiments, all algorithms required to explore less than 10% of all configurations. The only cases that required to explore more than 10% of configurations were experiments *DCN81* and *DVN82*, where the number of optimal configuration to be found was less than 2%. Fig. 4.8 emphasizes that all proposed strategies performed similarly, with the exception of *n-CPT sensitivity analysis*, which did not show a degradation in the cases of very few optimal configurations and required to explore a significantly lower number of configurations.

The values in Table 4.3 and in Fig. 4.8 concern the absolute number of steps required, and configurations explored, to find one of the optimal configurations among the total amount of 256 configurations. They provide an overview of the behavior of the strategies proposed in this chapter in problems of different difficulty with a search space of 256 possible solutions. Fig. 4.9 compares the percentage of explored configurations by the different strategies with the percentage of optimal configurations to be found.



(a) *uniform*

(b) *mostly compliant*

(c) *mostly violating*

Figure 4.8: *Average percentage of explored configurations before finding an optimal one.*

Note that, in problems with more than 6% of optimal configuration, the strategies did not exhibit significant differences. In more difficult problems (less than 3% of optimal configurations), the number of configurations to explore increased up to 20% with *Synergy+MLE*, *Combined synergy* and in particular with *Naive sensitivity analysis*. *Naive synergy* and *State-based*, instead, as reported above, exhibited a similar behavior in most of the cases. In problems with less than 2% of optimal configurations, however, they also required to explore a higher number (up to ∼15%) of configurations. Finally, the figure shows the robustness of *n-CPT sensitivity analysis*: despite performing slightly worse than other strategies in some experiments, *n-CPT sensitivity analysis* never required to explore more than 5% of all configurations, even in problems with about 1% of optimal configurations.

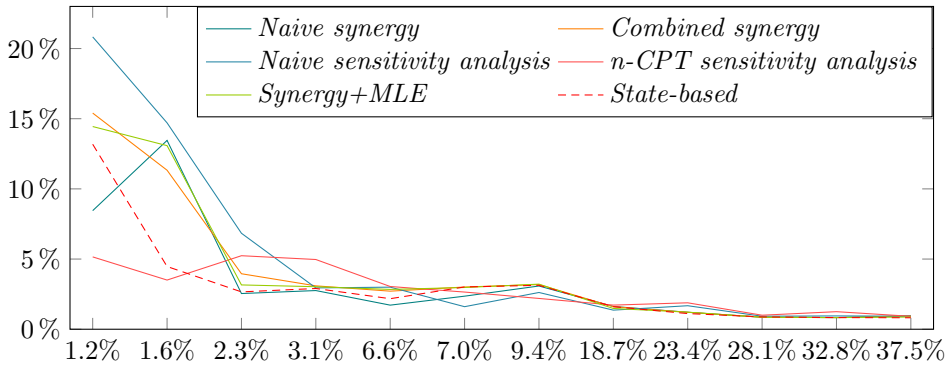


Figure 4.9: *Average percentage of explored configurations (y-axis) compared to the percentage of optimal configurations in the 12 different experiments (x-axis).*

4.7 Discussion

The results reported in Sec. 4.6 show that our proposed strategies can be employed to effectively revise at run-time the sanctions of the enforced norms to quickly improve the performance of the system (in terms of achievement of the system-level objectives). In particular, on 12 problems of different difficulty, our strategies reached optimal system's configurations after very few norm revisions. Starting with no initial knowledge about the effectiveness of the possible configurations, all the strategies explored on average less than 10% of all possible configurations before finding an optimal one. In the simplest experiment (*DVN32*), all strategies required to explore on average less than 1% of all possible configurations. In the same experiment, an uninformed strategy that does not consider run-time information and *randomly* tries a new configuration when the current one is not optimal would explore, on average, 62.5% of the configurations. In the most difficult experiment (*DVN82*), while a *random* strategy would explore on average 98.8% of the configurations to find one of the 1.2% optimal ones, our best performing strategy *n-CPT sensitivity analysis* explored, on average, only 5% of all possible configurations.

Our experiments identified three best-performing strategies: *Naive synergy*, *State-based* and *n-CPT sensitivity analysis*. We discuss each of these strategies and interpret the results and the conditions for their applicability.

Naive synergy determines, for each of the enforced norms, what type of synergy exists between the norm and the system-level objectives. Based on the identified synergy, *Naive synergy* increases or decreases the sanction for violating the norm. This strategy suits well cases where the observed data from MAS execution clearly highlights that a norm is better when either obeyed or violated. In experiment *DUN81*, for instance, in both contexts **c2** and **c3** the speed limit norm is effective only when fully obeyed by all agents (i.e., system configurations where some agents violate the speed limit are not optimal). In such experiment, and also in similar experiments such as *DUN82* and *DCN82*, the results confirmed that *Naive synergy* outperforms the other strategies.

The *State-based* strategy extends *Combined synergy*. Just like the latter, it considers the synergy between norms and objectives. Unlike *Combined synergy*, it also considers the most likely explanation for the objectives being not achieved. Furthermore, *State-based* takes also into account the global state of the system (the average norm obedience and objectives achievement) and suggests to revise only a certain type of norms at every iteration. This strategy is suitable for cases where many norms are enforced and where the obedience of agents to a norm is likely to affect also the obedience to other norms. In our experiments, *State-based* performed well in most of the cases, with the exception of the most difficult ones *DCN81* and *DVN82*, where, similarly to *Naive synergy* it required a higher number of revisions.

Note that, in experiments *DCN81* and *DVN82*, the optimal configurations are only 4 and 3, respectively, out of 256. To find the few optimal configurations quickly, it is necessary to have a strategy that precisely directs the norm revision. For this reason, synergy-based or category-based strategies, which only provide a direction for the revision (i.e., they simply suggest to either increase or decrease violations), were not the best in these experiments.

n-CPT sensitivity analysis, instead, provides a quantitative measure of how much change in the violations of each norm is required. This strategy is more precise, and, although it performed slightly worse than other strategies in a few cases, it showed a consistent convergence speed in all the experiments, including complex ones such as *DCN81* and *DVN82*. Thus, this strategy proved to be the most robust in terms of convergence speed. It is worth noting, however, that in cases where the desired achievement of the system-level objectives is not particularly restrictive and where many norms are enforced, *n-CPT sensitivity analysis* may be less adequate due to the higher computational effort it requires, especially if compared to simpler strategies like *Naive synergy*.

The worst-performing strategy, on average, is *Naive sensitivity analysis*. This strategy performed particularly bad (compared to the others) especially in the most difficult experiments, where, as explained above, very few configurations were needed to be found. This result, which may seem surprising since sensitivity-based strategies are generally more precise than the others, can be explained by the *naive* approach of the strategy in determining the amount of change in the violations of norms that is required to achieve the system-level objectives. In doing so, unlike *n-CPT sensitivity analysis*, this strategy considers the changes for only one norm at a time, assuming that the other parameters of the Bayesian Network (i.e., the amount of violations of other norms) would not change. After providing a suggestion, however, the strategy applies a sanctions revision to all norms together (i.e., it changes all the parameters of the network together), creating a discrepancy between the way the suggestions are provided and the implementation of such suggestions. This discrepancy appears evident in cases where the precision of the suggestions is essential to identify one of the few optimal solutions (e.g., *DCN81* and *DVN82*). Note, however, that all the proposed strategies are heuristics. Therefore, there is no guarantee that one strategy will always perform better or worse than the others. This is visible in the results: every strategy that we proposed, including *Naive sensitivity analysis*, performed better than the others in at least one experiment.

4.7.1 Limitations and Possible Extensions

In the following, we provide a discussion of some of the limitations and assumptions related to our framework and to the revision strategies that we proposed, outlining some possible future directions.

Preferences Changing over Time and Context

We considered agents with same preferences in all operating contexts. This simplification does not affect the generality of our approach. Our framework supports agents with different preferences in multiple operating contexts. In Sec. 4.4.2, we have shown how to use the estimation of the preferences of agents to determine an upper bound of the probability of violating a norm. In Sec. 4.5, we used such upper bound to guide the revision of the sanctions of the enforced norms in a certain operating context c. In order to use different preferences in varying operating contexts, it is possible to explicitly model the different contexts (as proposed, for example, in context-aware systems such as Ambient Intelligence systems [248]), and use an adequate upper bound in each of them.

This is made possible by the assumption that the preferences of agents (and therefore our estimation) do not change over time, i.e., we assumed that the behavior of the agent is consistent over time. We did not study the case of preferences changing over time. Preferences may change over time due to external factors inducing changes in the end-user's preferences, the introduction of new norms in the MAS, or changes in agents' own evaluation of states of affairs due to the acquisition of new experience [217, 308].

To support preferences that change over time, our framework needs to be adapted in a number of ways, briefly listed below. First, depending on the type of system, mechanisms for the dynamic elicitation of preferences should be employed and the estimation of the preferences should be dynamically replaced or updated (see, for example, mechanisms to learn and update dynamic preferences [96, 272]). Given the new preferences, the upper bound of the probability of violating a norm should be recomputed. System configurations that are ineffective when certain behaviors are exhibited by the agents, may be instead effective when different behaviors are exhibited, and vice-versa. When the preferences of the agents are changed, therefore, the knowledge acquired during the norm revision process about the effectiveness of the norms and about the relationship between norm violation and system-level objectives should be reconsidered and opportunely weighted. If the preferences of the agents change very quickly and repeatedly over time, the use of a static *Norm Bayesian Network* as the one described in Sec. 4.4 may be unfavourable and the use of different more dynamic learning techniques, e.g., Dynamic Bayesian Networks [240], may be necessary. Supporting partial and inaccurate preferences of agents, as briefly discussed in Sec. 4.7.1, could also help to cope with preferences changing over time.

Partial or Inaccurate Information

When looking for a new sanction set, we assumed not to have any knowledge about the norm violations that will be actually exhibited when a never-tried-before sanction

set is used to enforce norms. To guide the norm revision, we used the upper bound of a norm violation, a "safe" estimation of the actual norm violation that will be exhibited by agents. To calculate such upper bound we assumed an accurate (i.e., perfect) estimation of the preferences of the agents concerning the aspects of the system we aim to regulate.

The advantage of having an accurate estimation of the preferences of the agents is that we can define an upper bound for the probability of violating a (well defined) norm that is not too coarse-grained (e.g., a trivial upper bound is obviously a probability of 1, but this provides little information). As shown in Sec. 4.6, such an estimation, combined with our revision strategies, allows us to efficiently revise ineffective norms.

In some MASs, however, it is not possible to ensure a correct estimation of the agents' preferences [137]. Extending our work to support partial and/or inaccurate information about the agents' preferences requires an in-depth investigation. Based on the amount and type of information available, the accuracy and usefulness of the upper bound could significantly change. For partial information (e.g., we know that an agent type prefers a state of affairs over another, but we do not have information about all possible comparisons of alternative states of affairs), it is still possible to estimate a possibly more coarse-grained upper bound. For example, a trivial estimation could be obtained by assuming that agents always prefer to violate the norms related to aspects for which we do not have information. Less trivial estimations could be obtained for example by approximating the complete preferences by expressing the uncertain information as a belief function and leveraging the rationality principles of the preferences [92]. The estimated upper bound could be then refined over time by monitoring the behavior (i.e., the number of violations) of the agents. In case of inaccurate information (e.g., some of the available information about the preferences of agents is wrong, or the information available is only obtained from statistical data about the behavior of typical agents, or by learning the preferences from observed agents' choices [137]), the estimation of the probability of violating a norm should be treated more as a prediction, rather than an upper bound. In this case, techniques such as Bayesian Optimization [257], which attempts to find the minimum value of an unknown function, could be used for selecting new sanction sets and to refine over time the current estimation.

Nevertheless, a correct estimation of the preferences of the agents, as used in this chapter, does not imply perfect revision strategies. This is because the trend of the upper bound may be different from the trend of the actual norm violation, which is unknown a priori. The consequence of this can be illustrated on the example of Fig. 4.10, which reports a comparison between an upper bound (red dashed line) of the probability of violating a norm N, and N's exhibited violation (blue solid line), w.r.t. the sanction associated to N. Suppose the current sanction for a norm N is 0, with an exhibited norm violation $P(N_{viol}) = 0.3$, and the employed revision algorithm (e.g., *Naive synergy*) suggests to *reduce* violations of N. Here, the only possible choice for Algorithm 4, which relies on the estimation of the upper bound of violating a norm, is to select sanction 4 as new sanction, since for all other sanctions the upper bound is higher than the currently exhibited norm violation. Although sanction 2 would also satisfy the suggestion, this will remain unknown until such sanction is tried. If

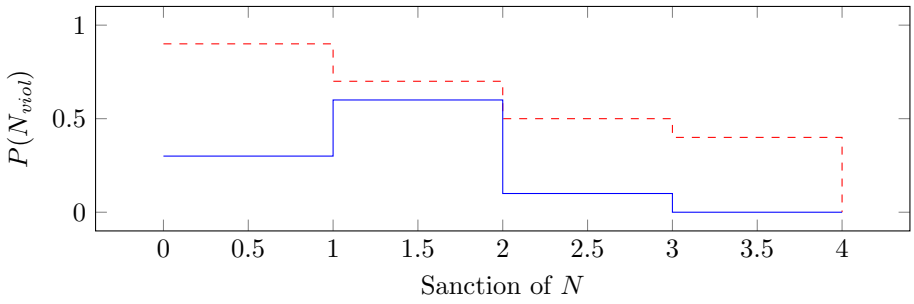


Figure 4.10: *Comparison between the upper bound (red dashed line) of the probability of violating a norm N, and N's exhibited violation (blue solid line), w.r.t. the sanction associated to N.*

the optimal value of $P(N_{viol})$ for the achievement of the system-level objectives is, for instance, around 0.1, our supervision framework will need to perform additional revision steps to select sanction 2.

Complexity of Preferences Representation

In this chapter, we introduced several types of preferences of rational agents as lists of tuples ordered according to different rational criteria. In our discussion and experiments, we considered complete preferences, i.e., we explicitly represented all possible alternative states of affairs. Such representation, however, grows exponentially with the number of norms and budgets. In real world scenarios, doing so may be possible only in restricted domains where the number of norms and the possible budgets of the agents is limited. In the general case, however, representing the complete preferences of agents may be infeasible. In this chapter, we attempted to lay down well founded principles for understanding the interplay between norms and the preferences of rational agents. For this reason, we provided a formal definition of different types of rational agents and we studied the properties of their preferences in relation with the chances to violate the enforced norms. We consider this as a necessary starting point for approaches to the run-time supervision of normative multi-agent systems involving rational agents. In Sec. 4.7.1, we outlined some guidelines for our framework to support also partial (and inaccurate) preferences, which is one obvious way to reduce the complexity of explicitly representing the complete preferences. We leave this as future work, together with the integration of automated preferences elicitation techniques within our framework.

Norms Importance

Our strategies do not make any distinction between norms: revisions are applied to all the norms. This approach can be extended to support a selective revision that takes into account of the importance of a certain norm for the achievement of the objectives. Consider the derivative in Eq. 4.7, which describes the impact of changes in $P(N_{viol})$ on $P(\mathbf{O}_{true})$ in a context $\mathbf{c}$. High values of such derivatives imply that changes in the

violations of norm N have bigger impact on $P(\mathbf{O}_{true})$. We call such derivative for a norm N the *importance* [288] *of norm* N in context $\mathbf{c}$. By computing the importance of all norms, we obtain an ordering between norms w.r.t. the system-level objectives. The strategies of Sec. 4.5.1 could be then applied to the k most important norms. Although there is no guarantee that this approach will be more effective, it applies to cases in which revising norms comes at a cost, and therefore minimizing the number of revisions is important.

In addition to the importance of a norm, the observed data from MAS execution allows to analyze the relationship between pairs of norms and to detect weather some of the following properties hold.

Additive synergy between two norms. This property, based on the concept of additive synergies in qualitative probabilistic networks [292], describes a situation where it is more likely to achieve the objectives when two norms are either both obeyed or both violated. Formally, two norms $N1$ and $N2$ exhibit an additive synergy when $P(\mathbf{O}_{true}|N1_{ob}N2_{ob}) + P(\mathbf{O}_{true}|N1_{viol}N2_{viol}) \geqslant P(\mathbf{O}_{true}|N1_{ob}N2_{viol}) + P(\mathbf{O}_{true}|N1_{viol}N2_{ob})$. The norms that exhibit an additive synergy with some of the k most important ones, could also be considered among the norms to be revised.

Product synergy between two norms. This property, based on the concept of product synergies in qualitative probabilistic networks [293], expresses how the value of one norm (e.g., $N1$ obeyed) influences the probability of the values of another norm (e.g., $N2$ obeyed), upon knowing the value for a common child (e.g., $\mathbf{O}$ true). For instance a negative product synergy says that observing $N1$ obeyed makes less likely to observe $N2$ being obeyed. Formally, two norms $N1$ and $N2$ exhibit a negative product synergy when $P(\mathbf{O}_{true}|N1_{ob}N2_{ob}) \cdot P(\mathbf{O}_{true}|N1_{viol}N2_{viol}) \geqslant (\leqslant) P(\mathbf{O}_{true}|N1_{ob}N2_{viol}) \cdot P(\mathbf{O}_{true}|N1_{viol}N2_{ob})$. This property can be used to choose between two norms to revise: it is enough to revise one of them to obtain an effect on the other.

Conflicting Norms

In this chapter, we assumed that the norms that are enforced are not conflicting, i.e., obeying a norm does not prevent *a priori* agents to obey other norms. This work focuses on regulative norms: norms enforced by an institution in order to regulate the behavior of the agents so to achieve desired system-level properties. In this context, we believe that an institution should not enforce conflicting norms, and we rely on normative conflict resolution mechanisms [281]. Despite this, our framework currently supports conflicting norms as long as the agents are aware of such conflicts, i.e., as long as the preferences of agents already take into account the conflicts. If two norms N_1 and N_2 are conflicting, obeying N_1 prevents the agents to obey N_2 and vice-versa. The preference of an agent that is aware of the conflict, determines whether the agent prefers to obey N_1 and pay a sanction for N_2, or vice-versa. This information is sufficient in our framework to estimate the upper bound for the violation of the norms and revise the sanctions of the norms when needed. Additionally, the information of the conflict could also be explicitly used to improve the performance of our revision

strategies, similarly to the use of the product synergies described in Sec. 4.7.1: if obeying a norm agents cannot obey another norm, then it is sufficient to revise one sanction to obtain an effect also on the violation of the other norm.

Neighborhood Expansion

When a norm revision is triggered, our supervision mechanism searches for a new sanction set that satisfies the suggestions provided by one of the heuristic strategies. The neighborhood of a configuration, in the current hill climbing implementation of the supervisor, is composed by exactly one sanction set (configuration): the one that best satisfies the suggestions. An immediate extension of this approach is to expand the neighborhood definition, by including not only the best satisfying configuration, but also sub-optimal ones: those configurations that "almost" satisfy the suggestions provided. This extension is easily supported by our supervisor, and it better fits the typical usage of the hill climbing optimization technique. By expanding the neighborhood, the number of revision steps required by the supervision mechanism to find an optimal configuration could possibly further decrease. The challenge in expanding the neighborhood is in appropriately defining almost-satisfying suggestions. Different distance metrics and criteria could be considered in order to do so. Adopting a neighborhood composed only by the best satisfying configuration allowed us, however, to analyze the quality of the suggestions provided by our algorithms without further overloading the experimentation with additional parameters. Experiments with different neighborhood definitions will be carried on in future work, considering also a bigger case study.

4.8 Conclusions

In a MAS, the complexity and unpredictability of the agent interactions and of the environment must be taken into account to maximize the achievement of the system-level objectives. When engineering MASs, the available knowledge of these dynamics is only partial and incomplete. As a consequence, MASs need to be supervised and regulated at run-time.

In this chapter, we proposed a supervision mechanism that relies on *norms with sanction* to influence agent behavior and regulate a MAS [68]. We considered MASs where agents are rational, i.e., they always choose to achieve their most preferred state of affairs. We characterized rational agents through their preferences and we made use of an estimation of the agents' preferences to guide the supervision of the MAS. Our mechanism automatically revises the sanctions that are employed to enforce the norms. To do so, it first interprets—through a Bayesian Network—observed data from MAS execution in terms of how well certain norms contribute to the achievement of the system-level objectives in different operating contexts. Then, it suggests how to revise the sanctions based on the knowledge learned at run-time and on the agents' preferences. We proposed six heuristics for the suggestion of sanction revisions.

An evaluation of the strategies through a traffic regulation simulation shows that our heuristics quickly identify optimal norm sets. We performed 12 different experiments on a ring-road traffic simulation, differing for the difficulty of the problem: the

number of optimal norm sets to be found among all the possible ones ranged from 1.2% to 37.5%. All the proposed strategies explored a small number of norm sets before finding an optimal one. In particular, the strategy *n-CPT sensitivity analysis*, based on the sensitivity analysis technique from probabilistic reasoning [76], on average never required to explore more than 5% of all possible norm sets in order to find one of the optimal ones.

This work paves the way for numerous future directions, some of which are sketched in Sec. 4.7.1. An in-depth evaluation of the scalability and computational complexity of the presented approach is necessary to assess its suitability for MASs with many norms and sanctions. Our simple language for representing norms and agents' preferences can be extended to consider complex norm types beyond atomic propositions. Our agent population was defined according to specific types. Future work should study the effect of agents that deviate from the prototypical agent types. Finally, in addition to the revision of the sanctions, the strategies here presented could be applied also to the revision of the norm proposition so to synthesize new norms. As a first step in this direction, in Chapter 5 we study how to revise and synthesize conditional norms based on execution data.

5 Data-Driven Revision of Conditional Norms with Deadlines

In multi-agent systems, norm enforcement is a mechanism to steer the behavior of individual agents so to achieve desired system-level objectives. As discussed in the previous chapters, however, due to the dynamics of multi-agent systems it is hard to design norms that guarantee the achievement of the system-level objectives in every operating context. Moreover, these objectives may change over time, thereby making previously defined norms ineffective. In this chapter, we investigate the use of system's execution data to automatically synthesise and revise conditional norms with deadlines, a type of behavior-based norms aimed at prohibiting or obliging agents to exhibit certain patterns of behaviors. This chapter aims at bridging the gaps in the literature concerning the lack of practical data-driven approaches for the synthesis and revision of behavior-based norms. In particular focuses on the alignment of norms with system-level objectives whose causal relation with the norms is not known and that are not directly enforceable on autonomous agents, e.g., an objective "no accidents shall occur in the highway". We present theoretical results on the complexity of norm synthesis and norm revision that call for approximate algorithms. We then propose an heuristic approach that determines approximate norm revisions. The approach consists of two steps: the synthesis step determines a set of possible revisions of the conditional norms applying heuristic operations on their components (i.e., condition, prohibited or obliged state, deadline); the selection step interprets norms as binary classifiers for execution traces, and chooses the final revised norms (classifiers) that perform better on the available execution traces. We evaluate our heuristics using a state-of-the-art, off-the-shelf, urban traffic simulator. The results show that our approach synthesizes revised norms that are significantly better aligned with the system-level objectives than the original norms.

This chapter has been submitted for journal publication and is currently under review as:

- Dell'Anna, Davide, Natasha Alechina, Brian Logan, Maarten Löffler, Fabiano Dalpiaz, and Mehdi Dastani. "Data-Driven Revision of Conditional Norms in Multi-Agent Systems".

This chapter provides an answer to research questions **RQ 3-4** from Chapter 1 in the context of synthesis and revision of conditional norms in MASs.

5.1 Introduction

In a multi-agent system (MAS), autonomous agents operate and interact in a shared environment [297]. To ensure that the emergent MAS behavior is aligned with some desired system-level objectives, the behavior of the agents should be monitored and coordinated [68]. For example, take a smart highway where the agents are autonomous vehicles whose speed and CO_2 emissions are monitored by sensors. The objectives of this smart highway may include ensuring safety, smooth traffic flow and low CO_2 emissions. To fulfill system-level objectives that may not be directly observable and controllable, without over-constraining the agents' autonomy, norms and their enforcement are widely adopted mechanisms [11]. For example, agents can be coordinated through traffic norms regarding maximum speed or minimum safety distance.

When designing a MAS, however, it is hard to specify norms that will guarantee the achievement of the system-level objectives (MAS objectives, from now on) in every operating context. Modeling all possible aspects of a MAS, including the dynamic interactions among agents as well as between agents and complex environments, is difficult if not impossible, unless one assumes that models of the agents and the environment are available [121]. Moreover, the MAS objectives may change over time, requiring also an adaptation of the norms that are enforced to guarantee such objectives. To cope with this issue and to keep the norms effective in achieving the MAS objectives, the enforced norms need to be continuously evaluated and possibly revised.

Several formal contributions to norm revision have been proposed, including logics for norm change [25, 180], design patterns for the iterative revision and verification of a specification [172], automated refinement of normative specifications via inductive logic programming [94], and runtime revision of sanctions based on agents' preferences [121]. However, most of the literature on norm revision relies on the knowledge of agents' internals and makes assumptions about the relationship between agents behavior, norms and MAS objectives in the operating environment.

In this chapter, we relax these assumptions. We treat agents as a black box, only assuming that they are norm-aware in the sense that they respond to the enforcement of a norm, and we consider the case when the knowledge about the relationship between agents behavior, norms and MAS objectives, relies only on the monitored system's execution.

We propose a general mechanism for norm revision that solely relies on the system's execution data: execution traces that describe the behavior exhibited by the agents and are labeled with a *boolean evaluation* of the MAS objectives. The proposed revision mechanism, which we call *DNR* (from Data-driven Norm Revision), is general in the sense that it does not rely on domain knowledge about the specific MAS objectives, but only uses the given boolean labeling of the execution traces. The labeling distinguishes two sets of execution traces: the *positive traces* set, including finite execution traces labeled as *true*, expressing the fact that the traces satisfy the

MAS objectives; and the *negative traces* set, including finite execution traces labeled as *false* (the trace does not satisfy the MAS objectives).

We focus on the revision of conditional norms with deadlines, a type of norms widely used in the normative MAS literature to express behavioral properties [10]. Conditional norms with deadlines are represented as tuples $(\phi_C, Z(\phi_Z), \phi_D)$, where $Z \in \{P, O\}$ indicates whether the norm is a prohibition (P) or an obligation (O) and ϕ_Z characterizes a state that is prohibited (or obliged) to occur after a state where the condition of the norm ϕ_C holds, and before a state where the deadline of the norm ϕ_D holds. Conditional norms can be used to distinguish finite execution traces as compliant or violating. The goal of our revision is to determine a set of norms that is *aligned* with the MAS objectives, i.e., that prohibits all negative behaviors/traces and permits all positive behaviors/traces.

The main contributions of this work are as follows.

- We formulate the revision of conditional norms with deadlines as a decision problem and demonstrate that this problem is NP-complete.
- Motivated by the complexity results, we propose *DNR*: a heuristic approach to obtain approximate revisions of the norms, i.e., revised norms that are not always perfectly aligned with the MAS objectives. *DNR* consists of two steps: the *synthesis step* determines a set of possible revisions of the conditional norms; the *selection step* chooses the final revised norms from the set of possible revisions based on statistical measures to be computed on the available execution traces.
- We report on the complexity of the approach and on its experimental evaluation on an agent-based traffic simulation.

Organization. Sec. 5.2 discusses related work. Sec. 5.3 provides the necessary background on normative MASs and conditional norms. Sec. 5.4 formalizes the notion of norm revision and presents theoretical results on its complexity. Sec. 5.5 describes our heuristic approach to cope with the high computational complexity of the problems. Sec. 5.6 reports our experimentation. Sec. 5.7 and Sec. 5.8 present threats to validity, conclusions, and future work.

5.2 Related Work

The synthesis and revision of norms has been studied from multiple points of view.

Alechina *et al.* [10] introduce the concept of norms approximation in the context of imperfect monitors. A norm is synthesized to approximate an original norm in order to maximize the number of violations that an imperfect monitor can detect. Although presented with a different goal in mind, this work inspires ours. We assume, however, perfectly monitorable norms, and we aim at synthesizing norms that are better aligned with the MAS objectives by using execution data.

Miralles *et al.* [208] present a framework for the adaptation of MAS regulations at runtime. They represent conditional norms via norm patterns and describe an adaptation mechanism based on case-based reasoning. Adaptation is performed at

runtime individually by a number of assistant agents and then, via a voting mechanism, a final adaptation is approved. The decision on how to adapt norms is taken based on similar previously seen cases. Dell'Anna *et al.* [121] propose a framework for the runtime selection of alternative norms based on runtime data and for the revision of the sanctions of norms based on the knowledge of agents preferences. Unlike these approaches, we do not assume knowledge of the agents' internals, e.g., their preferences or their reasoning and communication capabilities.

Corapi *et al.* [94] and Athakravi *et al.* [24] discuss the application of Inductive Logic Programming (ILP) [128] to norm synthesis and norm revision. In their work, the desired properties of the the system are described through use cases (event traces associated to a desired outcome state). Given the use cases, the authors propose to use ILP to revise the current norms so to satisfy the use-cases. In such approach, norms and desired outcome are strictly coupled: the desired outcomes of execution traces are expressed in the same language of the norms and, therefore, are directly enforceable. So-called mode declarations are used as heuristics to drive the revision. In our approach we consider a type of desired MAS objectives that cannot be directly enforced, and we use norms as a means to achieve such objectives (e.g., a speed limit norm is a means to achieve vehicles' safety, but it is not possible to directly enforce safety on vehicles: "no accidents should occur" is not directly enforceable on drivers). In our work, the only knowledge of the MAS objectives available to the revision mechanism, is a given boolean labeling of the execution traces. The causal relation between norms and MAS objectives is not given. Because we do not assume that the underlying causal structure of the domain is known to our revision mechanism, we are unable to generate provably correct norm revisions as in ILP-based approaches. This is why we use statistical analysis to drive the revision of norms. ILP-based approaches and our approach can therefore be seen as representing different trade-offs between the amount of background knowledge assumed about the possible causes of norm violations, and the guarantees that can be given regarding a particular (candidate) revision.

Christelis *et al.* [82] devise algorithms that introduce prohibitions in a MAS by setting preconditions to the actions the agents can perform in a regimentation setting. In our work, we do not assume that regimentation is available.

Lion [210] is an algorithm for the synthesis of liberal normative systems, i.e., an algorithm that synthesises norms trying to set as few constraints as possible on the agents' actions. To guide the synthesis process, the authors make use of a normative network: a graph structure that characterizes the generalization relationship between different norms. They use such graph to synthesize more general, that is more liberal, norms when possible. The norms synthesized by Lion are so-called action-based norms, which prohibit agents to perform actions in certain states [12]. In our work, we focus on the problem of revising conditional norms with deadlines, which are behavior-based, or path-based, norms, prohibiting (or obliging) agents to exhibit certain behaviors. Furthermore, differently from Lion, where the norms are synthesized so to achieve certain properties of the normative systems (such as its liberality), our norm revision is meant to align the enforced norms with MAS objectives, which are properties that are desired from the behavior of the whole MAS. We consider the liberality aspects of the norms an interesting possible extension of our

work that could be integrated as a criterion when selecting a new norm among the possible revisions in the *selection step*.

The concept of generalization of a norm described by Morales *et al.* [210] also relates to our operations of weakening and strengthening of a norm. Weakening a norm (referred to as relaxing a norm, in the previous chapters) generates more general norms, while strengthening generates more specialized ones. Similar operations have been described also in the software engineering literature, for example by Kafali *et al.* [172], who define design patterns for the iterative revision and verification of a specification. We propose an heuristic to perform such operations for conditional norms with deadlines.

Game theoretic concepts have been employed to guide norm synthesis, e.g., in the work by [212, 213]. Their control loop includes game recognition, payoff learning, and norm replication. Unlike us, they focus on the goals of the individual agents, while in our setting, we concentrate on MAS objectives. Furthermore, Bulling *et al.* [68] have used concurrent game structures to illustrate how the enforcement of norms can change agent behavior via regimentation and sanctions. Their work shows the high computational complexity of reaching a Nash equilibrium mapping, thereby providing evidence for our heuristic operations.

Mahmoud *et al.* [200] propose an algorithm for mining regulative norms that identifies recommendations, obligations, and prohibitions by analyzing events that trigger rewards and penalties. They focus on agents joining an open MAS who have to learn the unstated norms; we, instead, study how to alter existing norms from the point of view of a centralized authority.

Our work is influenced by research on norm change, including logics for norm change [25, 180], the study of the legal effects of norm change, analyzed and formalized by Governatori and Rotolo [156], and the contextualization of norms [167], which studies how to refine norms to make them suitable for specific contexts. In our framework, this corresponds to modifying the detachment condition and the deadline of the norms.

5.3 Normative Multi-Agent Systems

Consider a *10 km* long highway section. The traffic authority aims at reducing the CO_2 emissions generated by vehicles driving through that section below a threshold value t. To do so, a norm has been defined and enforced: *each vehicle entering the 2nd km of the highway is prohibited to drive faster than 70 km/h until it reaches the 7th km.* After monitoring the highway under the enforcement of such norm, the traffic authority discovers that, although the CO_2 emissions decreased, the highway throughput was significantly reduced. The traffic authority would like a tool to automatically revise the enforced norm so that the throughput of the highway is increased as much as possible, while keeping CO_2 emissions below t.

The highway segment is a simple example of a normative MAS. Vehicles are autonomous agents, each instantiating a certain agent type that characterizes its behavior on the highway. For instance a *car* agent type can reach a maximum speed of 200 km/h with a maximum acceleration of $2.9 m/s^2$, while a *truck* agent type can

drive up to 130 km/h, with a maximum acceleration of $1.3m/s^2$. To achieve the MAS objectives of reducing CO_2 emissions while maximizing highway throughput, agents' behavior is regulated by enforcing norms concerning the speed limits.

5.3.1 Agent Behavior and Conditional Norms

We assume that the behavior exhibited by the agents in the MAS is represented by a set of finite traces Γ, one trace per agent. Each trace consists of a finite sequence of states, where each state is labelled with propositional variables from a propositional language L. We further assume the existence of an initial set of norms[1] to be revised: we consider *conditional prohibitions*, and in the remainder of this chapter we use the terms *norm* and *prohibition* interchangeably. We omit an analogous discussion also for conditional obligations, and we report in the appendix the complexity results and the heuristic operations for obligation.

Definition 15 (Conditional Prohibition)**.** *Let ϕ_C, ϕ_P and ϕ_D be boolean combinations of propositional variables from a propositional language L. A conditional prohibition is a tuple $(\phi_C, P(\phi_P), \phi_D)$. A conditional prohibition is violated on a trace $(s_1, s_2, \ldots, s_m)$ if there are i, j with $1 \leqslant i \leqslant j \leqslant m$ such that ϕ_C is true at s_i, ϕ_P is true at s_j, and there is no k with $i < k < j$ such that ϕ_D is true at s_k.*

A conditional prohibition $(\phi_C, P(\phi_P), \phi_D)$ is *detached* in a state where the condition ϕ_C holds, and detachment persists until the norm is obeyed or violated. A prohibition is *obeyed* on a trace if it is either not detached on the trace, or no state satisfying ϕ_P is encountered before the execution reaches a state satisfying ϕ_D. A prohibition is *violated* if it is detached and a state satisfying ϕ_P is encountered before execution reaches a state satisfying ϕ_D. Conditional norms can be evaluated on finite traces in linear time [10].

The norm *each vehicle entering the 2nd km of the highway is prohibited from driving faster than 70 km/h until it reaches the 7th km* from the abovementioned example, for instance, can be represented as a conditional prohibition $(km_2 \wedge (car \vee truck), P(sp_{70}), km_7)$. The components $\phi_C = km_2 \wedge (car \vee truck)$, $\phi_P = sp_{70}$ and $\phi_D = km_7$ are boolean combinations of propositional variables from a propositional language $L = VP \cup VS \cup VT$, where $VP = \{km_1, \ldots, km_{10}\}$ is the subset of the language referring to vehicle positions on the highway, $VS = \{sp_x \mid 1 \leqslant x \leqslant 220 \,\&\, x \in \mathbb{N}\}$ denotes vehicle speeds in km/h, and $VT = \{truck, car\}$ represents vehicle types.

We also assume a function that is able to label each trace with a *single boolean evaluation* of the MAS objectives that the norms are intended to bring about. Such function partitions the set Γ into Γ_T (positive traces) and Γ_F (negative traces), depending on whether the traces are labeled as *true* (expressing the fact that they satisfy the MAS objectives) or as *false* (they do not satisfy the MAS objectives), respectively.

In our example, traces are logs of car journeys from the beginning to the end of the highway section, and the MAS objectives regard CO_2 emissions and traffic

[1] We consider sets of norms that are conflict-free, in the sense that obeying one norm does not prevent, a-priori, agents from obeying other norms. We focus on norms that are enforced by an institution in order to regulate the behavior of the agents so to achieve the desired MAS objectives. In this context, we believe that an institution should not enforce conflicting norms, and we assume that normative conflict resolution mechanisms [281] are adopted.

Table 5.1: *Example of dataset* Γ *with one norm* n*.*

Trace	n	Objectives	Type
ρ_1	obeyed	true	True Positive
ρ_2	obeyed	false	False Positive
ρ_3	violated	true	False Negative
ρ_4	violated	false	True Negative

throughput. Norms could prohibit exceeding a speed limit or the use of certain types of vehicles, while the MAS objectives could be evaluated based on the emissions generated by the journey and the time taken, and classified as positive or negative. A partial example of log is shown in Table 5.1 for the case of one norm.

We distinguish four cases as reported in the fourth column of Table 5.1.

- Trace ρ_1 describes a behavior that is currently permitted by norm n (the trace *obeys* n) and it is a positive trace according to the MAS objectives (it is labeled *true*). Borrowing terminology from statistics, we call ρ_1 a True Positive, to indicate that the trace is correctly (w.r.t. the MAS objectives boolean evaluation) permitted by n.

- Trace ρ_2 describes a behavior that is currently permitted by norm n but should be prohibited, since ρ_2 is a negative trace according to the MAS objectives (label *false*). We call ρ_2 a False Positive, to indicate that the trace is erroneously permitted by n.

- Trace ρ_3 describes a behavior that is currently prohibited by n but should be permitted, since ρ_3 is a positive trace. We call ρ_3 False Negative, to indicate that the trace is erroneously prohibited by n.

- Finally, ρ_4 describes a behavior that is currently prohibited by n and is a negative trace. We call ρ_4 a True Negative, for the trace is correctly prohibited by n.

In this work, we discuss the revision of a given set of norms so to align them with the MAS objectives, or, in other words, so that false negative behaviors are no longer prohibited and false positive ones are no longer allowed.

5.4 On the Complexity of Norm Revision

We start this section with describing the problem of *synthesising* a norm (prohibition or obligation) given a set of traces, such that it is aligned with the MAS objectives. We then establish the complexity of this problem and use it to obtain complexity results for *norm revision*: modifying an existing conditional norm, and synthesising or revising a set of conditional norms.

In the most general case, there is no conditional norm to start with, and we are given a set of positive and negative (with respect to satisfying the system objective) traces and need to synthesise the conditional norm. Clearly, this is not always possible;

two sets of traces may not be distinguishable by a single conditional norm, or even by a set of conditional norms. A simple abstract example is given below:

negative trace: (s_1, s_1, s_2, s_3)

positive trace: (s_1, s_2, s_3)

The two traces above cannot be distinguished by a conditional norm (nor by a set of conditional norms).

Next we define formally the decision problem we call *prohibition synthesis*.

Definition 16. *The* prohibition synthesis problem *is the following decision problem:*

Instance *A finite set of observable properties of states $p_1, \ldots, p_n$; a finite set Γ of finite traces partitioned into Γ_T (positive traces) and Γ_F (negative traces), where each trace is given as a sequence of state descriptions (conjunction of literals built from $p_1, \ldots, p_n$);*

Question *Are there three propositional formulas ϕ_C, ϕ_P, and ϕ_D built from $p_1, \ldots, p_n$ such that*

Neg *every trace in Γ_F violates $(\phi_C, P(\phi_P), \phi_D)$*

Pos *no trace in Γ_T violates $(\phi_C, P(\phi_P), \phi_D)$*

When generating formulas for components in the conditional norm, we can assume that they are in disjunctive normal form (disjunctions of state descriptions). A disjunction of state description corresponds to a set of states (states whose state description is one of the disjuncts). A state satisfies a disjunction of state descriptions if, and only if, its description is one of the disjuncts. If such a disjunction only contains state descriptions occurring in the input, then the size of formulas is linear in the size of the input.

Let S be the set of states occurring in Γ. Let for any ϕ built from $p_1, \ldots, p_n$, $\|\phi\| \subseteq S$ be the set of states where ϕ is true. Let $\rho[i]$ be the *i-th* state description in a trace ρ. The considerations above allow us to restate the prohibition synthesis problem as follows: given a set of positive traces Γ_T and negative traces Γ_F, find three sets of states X_C, X_P, X_D such that:

Neg For every trace $\rho \in \Gamma_F$, exist i and j with $i \leqslant j$ such that $\rho[i] \in X_C$, $\rho[j] \in X_P$, and there is no k with $i < k < j$ such that $\rho[k] \in X_D$.

Pos For every trace $\rho \in \Gamma_T$, if for some i and j, $i \leqslant j$, $\rho[i] \in X_C$, $\rho[j] \in X_P$, then there exists k such that $i < k < j$ and $\rho[k] \in X_D$.

Theorem 1. *The prohibition synthesis problem is NP-complete.*

Proof. The prohibition synthesis problem is clearly in NP (a non-deterministic Turing machine can guess the sets and check in polynomial time that they satisfy the conditions). To prove that it is NP-hard, we reduce 3-SAT (satisfiability of a set of clauses with 3 literals) to prohibition synthesis.

Suppose a set of clauses $C_1, \ldots, C_n$ over variables $x_1, \ldots, x_m$ is given. We generate an instance of the prohibition synthesis problem such that it has a solution iff

$C_1, \ldots, C_n$ are satisfiable (each clause contains at least one true literal). The set of states in the prohibition synthesis problem consists of two states s and t which are a technical device, and for each variable x_i, two states u_i and v_i, intuitively meaning that x_i is true (u_i) or false (v_i).

The set of negative traces Γ_F contains:

- a two state trace (s, t) [together with $s, t \notin X_C \cap X_P$ below, this forces $s \in X_C$ and $t \in X_P$]
- for every variable x_i in the input, a trace (s, v_i, t, s, u_i, t) [this ensures that either v_i or u_i are not in X_D]

The set of positive traces Γ_T contains:

- a single state trace (s) [so s cannot be in $X_C \cap X_P$];
- (t) [so t cannot be in $X_C \cap X_P$];
- for every variable x_i in the input: (s, v_i, u_i, t) [this means that either v_i or u_i are in X_D]; (v_i); (u_i); (v_i, t); (u_i, t); (s, v_i); (s, u_i);
- for every pair of variables x_i, x_j in the input: (v_i, u_j); (u_j, v_i) [the last eight types of traces ensure that v_i and u_i are not in X_C or X_P];
- for each clause C in the input over variables x_j, x_k, x_l: (s, z_j, z_k, z_l, t) where z_i is u_i if x_i occurs in C positively, and v_i if it occurs negatively.

The reduction is polynomial in the number of variables (quadratic) and clauses (linear). We claim that there exists an assignment f of $0, 1$ to $x_1, \ldots, x_m$ satisfying $C_1, \ldots, C_n$ if, and only if, there is a solution to the prohibition synthesis problem above where $X_C = \{s\}$, $X_P = \{t\}$, and for every i, $u_i \in X_D$ iff $f(x_i) = 1$ and $v_i \in X_D$ iff $f(x_i) = 0$.

Assume an assignment f satisfying $C_1, \ldots, C_n$ exists. Let $X_C = \{s\}$ and $X_P = \{t\}$. For every i, place u_i in X_D if $f(x_i) = 1$ and $v_i \in X_d$ iff $f(x_i) = 0$. This produces a solution because: s, t satisfies **Neg**; for every i, either u_i or v_i are not in X_D, so s, v_i, t, s, u_i, t satisfies **Neg**. Positive traces satisfy **Pos**: either s followed by t does not occur on a trace, or u_i, v_i occur between s and t and one of them is in X_D, or (from the clause encoding) one of the literals in the clause is true, so for positive x_i it means that u_i is in X_D and **Pos** is satisfied, or for negative $\neg x_i$ it means that v_i is in X_D and again **Pos** is satisfied.

Assume there is a solution to the prohibition synthesis problem. It is clear (see the comments next to traces) that it has to be of the form $X_C = \{s\}$, $X_P = \{t\}$ and X_D containing some u_is and v_is. In particular, since (s, v_i, u_i, t) is a positive trace, for every i either u_i or v_i has to be not in X_D. Set $f(x_i)$ to be 1 if u_i in X_D and 0 otherwise. Then each clause $C = \{\sim x_j, \sim x_k, \sim x_l\}$ is satisfied by f since for every clause there will be one literal which is true. This is because (s, z_j, z_k, z_l, t) is a positive trace, and either for some positive literal x_i, u_i is in X_D, or for some negative literal $\neg x_i$, v_i is in X_d, so u_i is not in X_D, so $f(\neg x_i) = 1$. □

The *obligation synthesis problem* can be stated similarly:

Definition 17. *The* obligation synthesis problem *is the following decision problem:*

Instance *A finite set of observable properties of states $p_1, \ldots, p_n$; a finite set Γ of finite traces partitioned into Γ_T (positive) and Γ_F (negative), where each trace is given as a sequence of state descriptions (conjunction of literals built from $p_1, \ldots, p_n$);*

Question *Are there three propositional formulas ϕ_C, ϕ_O, and ϕ_D built from $p_1, \ldots, p_n$ such that*

Neg *every trace in Γ_F violates $(\phi_C, O(\phi_O), \phi_D)$*

Pos *no trace in Γ_T violates $(\phi_C, O(\phi_O), \phi_D)$*

Theorem 2. *The obligation synthesis problem is NP-complete.*

The proof is given in Appendix C.

Next we consider the problems of (minimally) revising, weakening or strengthening conditional prohibitions (we omit the treatment of obligations, which is analogous).

Assume we are given a set of traces and a conditional prohibition $(\phi_C, P(\phi_P), \phi_D)$, and need to revise it so that it classifies the traces correctly. By revising we mean changing the original prohibition in a minimal way. The editing distance between conditional prohibitions can be defined in various ways. For example, for formulas ϕ_C, ϕ_P, ϕ_D in disjunctive normal form, this could be the sum of the numbers of added disjuncts and removed disjuncts for all three formulas. Or, we could represent each state description in each disjunct as a string of 0s and 1s and compare the number of flipped values. Note however that the set of non-equivalent propositional formulas built from the set $Q = \{p_1, \ldots, p_n\}$ is finite, and so is the number of possible different conditional prohibitions (or obligations). Regardless of how the distance between different conditional norms is defined, for a fixed set of propositional variables Q there is a maximal distance $max(Q)$ between any two norms using formulas built from Q. For example, let the distance between $(\phi_C, P(\phi_P), \phi_D)$ and $(\phi'_C, P(\phi'_P), \phi'_D)$ be $\Sigma_{X \in \{C,P,D\}} \mathit{diff}(\phi_X, \phi'_X)$, where $\mathit{diff}(\phi_X, \phi'_X)$ is the number of disjuncts the two formulas differ on: $\mathit{diff}((\phi_X, \phi'_X) = |\mathit{disjuncts}(\phi_X) \backslash \mathit{disjuncts}(\phi'_X)| + |\mathit{disjuncts}(\phi'_X) \backslash \mathit{disjuncts}(\phi_X)|$ (and $\mathit{disjuncts}(\phi)$ for a formula ϕ in disjunctive normal form is the set of disjuncts in ϕ). Then if $|Q| = n$, the largest possible value for $\mathit{diff}(\phi_X - \phi'_X)$ is 2^n, and the largest possible value for $max(Q)$ is $3 \cdot 2^n$.

Below we state the decision problem for minimal revision (asking whether there is a revision of distance less than m). It assumes some distance measure *dist* defined for any two conditional prohibitions α_1 and α_2 built over the same set of propositional variables Q.

Definition 18. *The (decision form) of the minimal prohibition revision problem is as follows:*

Instance *A number m; a finite set of observable properties of states $p_1, \ldots, p_n$; a finite set Γ of finite traces partitioned into Γ_T (positive) and Γ_F (negative); a conditional prohibition $(\phi_C, P(\phi_P), \phi_D)$.*

Question *Are there three propositional formulas ϕ'_C, ϕ'_P, and ϕ'_D built from $p_1, \ldots, p_n$ such that*

Dist $dist((\phi_C, P(\phi_P), \phi_D), (\phi'_C, P(\phi'_P), \phi'_D)) \leqslant m$

Neg *every trace in Γ_F violates $(\phi'_C, P(\phi'_P), \phi'_D)$*

Pos *no trace in Γ_T violates $(\phi'_C, P(\phi'_P), \phi'_D)$*

Theorem 3. *Let $dist(\alpha_1, \alpha_2)$ be computable in time polynomial in the size of α_1 and α_2, and the range of dist over norms built over propositions from Q be bounded by $max(Q)$. Then the minimal prohibition revision problem is NP-complete.*

Proof. The membership in NP follows from the fact that a solution can be guessed and checked in polynomial time.

NP-hardness is by reduction from the prohibition synthesis problem. Note that if a solution to the prohibition synthesis problem exists, it will be at most at distance $max(Q)$ from the input norm. So to solve the synthesis problem, we can ask for a solution to the minimal prohibition revision problem with $m = max(Q)$.

□

The decision problem for whether a prohibition can be weakened so that it no longer applies to a set of 'positive' traces can be stated as in Def. 19. By weakening, we mean that the prohibition is made less strict, that is, one or more of the following hold: the detachment condition is made harder to satisfy, the prohibited state itself is made harder to satisfy, the deadline is made easier to satisfy.

Definition 19 (Prohibition weakening problem)**.** *The* prohibition weakening problem *is the following decision problem:*

Instance *A finite set of observable properties of states $p_1, \ldots, p_n$; a finite set Γ of finite traces partitioned in Γ_F and Γ_T, where each trace is given as a sequence of state descriptions (conjunction of literals built from $p_1, \ldots, p_n$); a conditional prohibition $(\phi_C, P(\phi_P), \phi_D)$ where ϕ_X, $X \in \{C, P, D\}$, are propositional formulas built from $p_1, \ldots, p_n$, and all of Γ_F traces and some of Γ_T traces violate $(\phi_C, P(\phi_P), \phi_D)$.*

Question *Are there ϕ'_C, ϕ'_P, and ϕ'_D built from $p_1, \ldots, p_n$ such that: $\phi'_C \models \phi_C$, $\phi'_P \models \phi_P$, $\phi_D \models \phi'_D$,*

no $\rho \in \Gamma_T$ violates $(\phi'_C, P(\phi'_P), \phi'_D)$

every $\rho \in \Gamma_F$ violates $(\phi'_C, P(\phi'_P), \phi'_D)$;

Theorem 4. *Prohibition weakening problem is NP-complete.*

Proof. The membership in NP follows from the fact that a solution can be guessed and checked in polynomial time.

NP-hardness can be shown by reducing the prohibition synthesis problem to the prohibition weakening problem where the input norm is $(\top, P(\top), \bot)$. □

In a similar way, the prohibition strengthening problem can be defined and shown to be NP-complete.

Finally, we briefly state the *multiple prohibitions weakening (MPW) problem*: given a set of conditional prohibitions $n_1, \ldots, n_m$, and a set of traces partitioned in Γ_T and Γ_F, where all traces in Γ_F violate at least one of the norms, and some traces in Γ_T violate some of the norms, find weakened forms $n'_1, \ldots, n'_m$ of $n_1, \ldots, n_m$, such that all traces in Γ_F still violate one of the norms, but no trace in Γ_T does.

Theorem 5. *The MPW problem is NP-complete.*

Proof. Membership of NP: a non-deterministic TM can guess $n'_1, \ldots, n'_m$ and check that they satisfy the conditions in polynomial time. To check that each trace in Γ_F violates at least one of n'_i, for each n'_i mark the traces in Γ_F that violate it. If all traces are marked, the condition holds. NP-hardness follows from NP-hardness for the (single) prohibition weakening problem. □

5.5 *DNR*: A Heuristic for Approximate Norm Revision

The results in Sec. 5.4 lead us to proposing *DNR*, a more practical approach to norm revision. The approach aims at determining approximate revisions that are *better*, even though not always perfectly, aligned with the MAS objectives. *DNR* consists of two steps. Consider, for simplicity one norm n. In Sec. 5.5.4, we will show how the approach generalizes to multiple norms.

- *Synthesis step.* This step aims at generating a set revisions of a norm n. We discuss this step in Sections 5.5.1 and 5.5.2, where first we describe a number of heuristic operations to revise the components of a conditional prohibition, and then we show how to combine and use such operations to generate the set of revised norms.
- *Selection step.* This step aims at selecting one norm from the set of revised norms obtained in the *synthesis step*. We describe this step in Sections 5.5.3 and 5.5.3, where we characterize the concept of *alignment* of a norm with the MAS objectives, by means of statistical metrics that can be calculated on the dataset of traces Γ. The selected norm will be most aligned with the MAS objectives.

In our examples, we consider conditional prohibitions $(\phi_C, P(\phi_P), \phi_D)$, such that ϕ_C, ϕ_P and ϕ_D are boolean combinations of propositions from the languages $L_C = VP \cup VT$, $L_P = VS \cup VT$, and $L_D = VP$, respectively, with VP, VT, VS defined as per Sec. 5.3.1. In order to illustrate our approach to norm revision, consider the conditional prohibition $n = ((km_1 \wedge truck) \vee (km_2 \wedge car), P((sp_{36} \wedge truck) \vee (sp_{54} \wedge car)), (km_{10} \wedge truck) \vee (km_{10} \wedge car))$, which forbids *trucks* to drive faster than *36 km/h* throughout the highway section, and *cars* to drive faster than *54 km/h* from km_2 to the end of the highway section.

The prohibition n is equivalent to the set of eight *sub-norms* reported in Table 5.2. A sub-norm is a norm such that each component is one of the disjuncts (i.e., a

conjunction of literals) from the corresponding component of the original norm. A prohibition n is violated by a trace ρ if and only if at least one sub-norm is violated by ρ.

Table 5.2: *The sub-norms of a conditional prohibition* $n = ((km_1 \wedge truck) \vee (km_2 \wedge car), P((sp_{36} \wedge truck) \vee (sp_{54} \wedge car)), (km_{10} \wedge truck) \vee (km_{10} \wedge car))$.

n_1	$(km_1 \wedge truck, P(sp_{36} \wedge truck), km_{10} \wedge truck)$
n_2	$(km_1 \wedge truck, P(sp_{36} \wedge truck), km_{10} \wedge car)$
n_3	$(km_1 \wedge truck, P(sp_{54} \wedge car), km_{10} \wedge truck)$
n_4	$(km_1 \wedge truck, P(sp_{54} \wedge car), km_{10} \wedge car)$
n_5	$(km_2 \wedge car, P(sp_{36} \wedge truck), km_{10} \wedge truck)$
n_6	$(km_2 \wedge car, P(sp_{36} \wedge truck), km_{10} \wedge car)$
n_7	$(km_2 \wedge car, P(sp_{54} \wedge car), km_{10} \wedge truck)$
n_8	$(km_2 \wedge car, P(sp_{54} \wedge car), km_{10} \wedge car)$

Some of the sub-norms of a norm, *in a certain domain* may be irrelevant, as they may refer to situations that can never occur. For instance, in our running example, neither the prohibition nor the deadline of sub-norm n_5 in Table 5.2 can ever be true after the sub-norm is detached, for they refer to states that can appear only on execution traces of *trucks*, but the sub-norm can be detached only on execution traces of *cars*.

Since in *DNR* the revision is based on the set of traces Γ, we do not need to generate all the (exponentially many) sub-norms. We can instead generate, in polynomial time, only the sub-norms that are relevant for the set of traces Γ. For each $t \in \Gamma$, we can determine the relevant sub-norms in linear time with respect to the size of the original norm. In particular, we can read the trace state by state and keep track of one of the disjuncts c in Φ_C and one of the disjuncts p in Φ_P that made the norm detached and violated in t, respectively. Once we find a state where one of the disjuncts d in Φ_D holds, we generate the sub-norm $(c, P(p), d)$; else, we ignore the trace. In doing so we implicitly discard all possibly irrelevant sub-norms since, as mentioned above, they refer to situations that can never occur.

Our approach for the *synthesis step* consists of revising the relevant sub-norms of a norm. In order to revise a norm n so to permit a (sub)set of traces that n currently erroneously prohibits (false negative traces, as per Sec. 5.3), we need to revise (some of) the sub-norms of n that are currently violated by at least one of the traces in Γ. Similarly, in order to revise a norm n so to prohibit a (sub)set of traces that currently erroneously obey n (false positive), we need to revise at least one of its sub-norms so that they prohibit the traces.

Notice that in the case of false negatives, completely removing the sub-norms (we will refer later to this as disabling the sub-norm) is a possible, even though *extreme*, type of revision, as it may lead to the undesired *side effect* of generating possibly too many new false positives, as too many behaviors may be permitted. Analogously, if we want to prohibit more traces, a trivial and extreme possibility is to revise a sub-norm so that *nothing* is permitted anymore. This, may lead to the undesired side effect of generating possibly too many new false negative, as too many behaviors may be prohibited.

In Sec. 5.5.1 we propose a number of heuristic operations that can be used to revise the components *of a sub-norm*, based on the set of traces Γ. In Sec. 5.5.2 then, we combine such operations to generate a set of new possible revised sub-norms. The revised sub-norms will possibly contain, as special case, also the extreme revisions above illustrated. In the following, for clarity, we focus on a single *sub-norm*, and, for simplicity, from now on, we use the term norm instead of sub-norm.

5.5.1 Heuristic Revision Operations

We introduce a number of heuristic operations that can be used to revise the three components of a prohibition with respect to a given dataset Γ.

As a running example, we use a conditional prohibition $n_1 = (km_1, P(sp_{36} \wedge truck), km_{10} \wedge truck)$. We use ϕ_C, ϕ_P, ϕ_D to refer to the condition, prohibition and deadline of n_1, respectively. Each operation takes as input the current component of the norm (e.g., its condition ϕ_C) and produces a set of revised components (e.g., a set $MSC(\phi_C)$ of new conditions that are all more specific than the current condition ϕ_C). In Sec. 5.5.2, we will see how to combine these operations to obtain different types of revisions of a norm. In the following, we use $prop(x)$ to indicate a function that returns the set of literals contained in its argument x.

Condition

Make condition ϕ_C more specific. This operation generates conditions *more specific* than the current one, i.e., it determines a set $MSC(\phi_C) = \{\phi'_C \mid \phi'_C \models \phi_C\}$. With a more specific condition, the norm can detach in fewer traces. Consider the set Γ_{n_1} of traces in Γ *violating* norm n_1. Each trace in Γ_{n_1} contains a non-empty set of states where all literals in ϕ_C hold and are followed by at least a state where ϕ_P holds; else, the trace would not have violated n_1. Let CS be the union of such set of states, for all traces in Γ_{n_1}. Let Φ be a set of possible conjunctions of literals from $(prop(CS) \cap L_C) \backslash prop(\phi_C)$ with the size bounded by a polynomial in $|prop(CS)|$. The construction of Φ is domain-specific and is discussed in Sec. 5.5.1.

We make the condition more specific by adding a conjunct from Φ to ϕ_C, so that $\phi'_C = \phi_C \wedge \phi$, with $\phi \in \Phi$. $MSC(\phi_C) = \{\phi_C \wedge \phi \mid \phi \in \Phi\}$ is the set of all possible conditions, more specific than ϕ_C, that can be obtained by means of the above described operation w.r.t. Γ.

Example. Revising the condition $\phi_C = (km_1)$ into $\phi'_C = (km_1) \wedge truck$, the new norm will only apply to trucks.

Finally, note that a more specific condition than ϕ_C is $\phi'_C = \bot$. A norm with condition $\phi'_C = \bot$ can never be detached, and corresponds therefore to a *disabled* norm.

Make condition ϕ_C less specific. This operation generates conditions *less specific* than the current one, i.e., it determines a set $LSC(\phi_C) = \{\phi'_C \mid \phi_C \models \phi'_C\}$. With a less specific condition, the norm can detach in more traces. The idea is to make ϕ'_C hold in states that precede states in which the prohibition holds in the traces that *obey* the current norm. Let Φ be a fixed size set of possible conjunctions of literals

from $prop(OPS) \cap L_C$, with OPS set of states where ϕ_P holds in the traces obeying n, if any. We make the condition less specific by adding a disjunct from Φ to ϕ_C, so that $\phi'_C = \phi_C \vee \phi$, with $\phi \in \Phi$. $LSC(\phi_C) = \{\phi_C \vee \phi \mid \phi \in \Phi\}$ is the set of all possible conditions, less specific than ϕ_C, that can be obtained by means of the above described operation w.r.t. Γ.

Example. Revising $\phi_C = km_1 \wedge truck$ into $\phi'_C = (km_1 \wedge truck) \vee (km_1 \wedge car)$, the new norm will apply not only to trucks but also to cars.

Prohibited State

Make the prohibited state ϕ_P more specific. This operation generates prohibited states *more specific* than the current one, i.e., it determines a set $MSP(\phi_P) = \{\phi'_P \mid \phi'_P \models \phi_P\}$. With a more specific prohibited state, the norm can be violated by fewer traces.

Each trace in Γ_{n_1} (i.e., traces from Γ which *violate* norm n_1) contains a non-empty set of states where all literals in ϕ_P hold. Let PS be the union of such states, for all traces in Γ_{n_1}. We can apply the same strategy described for making the condition more specific, adding a conjunct that restricts the set of states that violate the norm.

We make the prohibited state more specific by adding a conjunct from Φ to ϕ_P, so that $\phi'_P = \phi_P \wedge \phi$, with $\phi \in \Phi$, and Φ defined analogously to the case of conditions but w.r.t. $(prop(PS) \cap L_P) \backslash prop(\phi_P)$, and same considerations on the number of conjunctions apply as above. $MSP(\phi_P) = \{\phi_P \wedge \phi \mid \phi \in \Phi\}$ is the set of all possible prohibited states, more specific than ϕ_P, that can be obtained by means of the above described operation w.r.t. Γ.

Example. Revising $\phi_P = (sp_{36} \wedge truck)$ into $\phi'_P = (sp_{36} \wedge truck) \wedge sp_{54}$, the new norm prohibits trucks to go faster than *54 km/h* instead of only *36 km/h*.

Similarly to the case of condition, a norm with prohibition $\phi'_P = \bot$ corresponds to a disabled norm, as it can never be violated in any trace.

Make the prohibited state ϕ_P less specific. This operation generates prohibited states *less specific* than the current one, i.e., determines a set $LSP(\phi_P) = \{\phi'_P \mid \phi_P \models \phi'_P\}$. With a less specific prohibited state, the norm can be violated by more traces. The idea is to make ϕ'_P true in states that are after states where the condition holds. Let Φ be a fixed size set of possible conjunctions of literals from $(prop(IPS) \cap L_P)$, with IPS set of states that are in between states where the condition and the deadline hold in the traces obeying n.

We make the prohibition less specific by adding a disjunct from Φ to ϕ_P, so that $\phi'_P = \phi_P \vee \phi$, with $\phi \in \Phi$. $LSP(\phi_P) = \{\phi_P \vee \phi \mid \phi \in \Phi\}$ is the set of all possible prohibited states, less specific than ϕ_P, that can be obtained by means of the above described operation w.r.t. Γ.

Example. Revising $\phi_P = sp_{54}$ into $\phi'_P = sp_{54} \vee sp_{36} = sp_{36}$, the new norm prohibits trucks to go faster than *36 km/h* instead of *54 km/h*.

Deadline

Make the deadline ϕ_D less specific. This operation generates deadlines *less specific* than the current one, i.e., it determines a set $LSD(\phi_D) = \{\phi'_D \mid \phi_D \models \phi'_D\}$. With a

less specific deadline, the norm can be violated by fewer traces. Since a conditional prohibition can be violated by a trace even if the deadline is not reached, the idea is to make ϕ'_D true in more states that occur before states where the prohibited state holds in the *violating* traces. Informally, we can say that we try to "insert" (when possible) the deadline before the prohibition is violated. Let CPS be the set of states of the traces in Γ_{n_1} that are in between states where the condition and the prohibition hold; and CPS_P the subset of CPS with states from the positive traces Γ_T. If $CPS_P = \varnothing$, no deadline revision can be applied. We can add to ϕ_D one (or more) disjuncts so that the new deadline is true in (some) state of CPS.

We make the prohibition less specific by adding to ϕ_D a disjunct from Φ (a fixed size set of possible conjunctions of literals from $prop(CPS) \cap L_D$ and analogous considerations on the number of disjunctions hold as above), so that $\phi'_D = \phi_D \vee \phi$, with $\phi \in \Phi$. $LSD(\phi_D) = \{\phi_D \vee \phi \mid \phi \in \Phi\}$ is the set of all possible deadlines, less specific than ϕ_D, that can be obtained by means of the above described operation w.r.t. Γ.

Example. Revising $\phi_D = (km_{10} \wedge truck)$ into $\phi'_D = (km_{10} \wedge truck) \vee (km_9 \wedge truck)$, we prohibit trucks to drive faster than *36 km/h* only until km_9, instead of km_{10}.

Make the deadline ϕ_D more specific. This operation generates deadlines *more specific* than the current one, i.e., it determines a set $MSD(\phi_D) = \{\phi'_D \mid \phi'_D \models \phi_D\}$. With a more specific deadline, the norm can be violated by more traces. The idea is to make ϕ'_D true in fewer states that are before states where the prohibited state holds. Let DS be the set of states DS where ϕ_D holds in the traces obeying n. Let Φ be a set of possible conjunctions of literals from $(prop(DS) \cap L_D) \backslash prop(\phi_D)$.

We make the deadline more specific by adding to ϕ_D a conjunct from Φ, so that $\phi'_D = \phi_D \wedge \phi$, with $\phi \in \Phi$. $MSD(\phi_D) = \{\phi_D \wedge \phi \mid \phi \in \Phi\}$ is the set of all possible deadlines, more specific than ϕ_D, that can be obtained by means of the above described operation w.r.t. Γ.

Example. Revising $\phi_D = (truck \wedge km_9)$ into $\phi'_D = truck \wedge km_9 \wedge km_{10}$, we prohibit trucks to drive faster than *36 km/h* for one more kilometer.

Table 5.3 summaries all the operations above described. Analogous operations can be devised also for the components of an obligation. We do not discuss them in details here, but we report their summary in Table 5.3.

The Set Φ

Several strategies could determine the set Φ, depending on available domain knowledge, the language L, available computational power, and personal preferences. For example, one may consider a criterion of minimality of the revision, that limits the number of literals in the conjunctions composing Φ. This can help obtaining revised components (and therefore norms) that are closer, so to say, to the current ones. Alternatively, or additionally, one may fix the maximum number of conjuncts in order to bound the computational time required to generate such set. In our experimentation, we use some domain knowledge to select only a fixed number of literals. For

instance, when we revise the speed limit, we consider only a fixed maximum number of higher or lower limits (the domain knowledge provides us with the partial ordering between the different literals representing the speed). This leads to a fixed number of conjunctions and also to revised norms that are closer to the original ones.

5.5.2 Revising the Norm, the *Synthesis Step*

We show here how to combine the operations from Sec. 5.5.1 in the *synthesis step* of our *DNR* approach to obtain different types of revised norms. Consider a norm $n = (\phi_C, P(\phi_P), \phi_D)$ to be revised. The set of all possible revisions of n that can be obtained by combining all the operations described above is the set $\mathcal{A}(n)$ defined as per Eq. 5.1, where the sets $MSx(\phi_x)$ and $LSx(\phi_x)$, for $x \in \{C, P, D\}$, are as per Sec. 5.5.1. Formally,

$$\mathcal{A}(n) = \{n' \mid n' = (\phi'_C, P(\phi'_P), \phi'_D) \;\&\; \phi'_C \in MSC(\phi_C) \cup LSC(\phi_C) \;\&\; \phi'_P \in MSP(\phi_P) \cup LSP(\phi_P) \;\&\; \phi'_D \in MSD(\phi_D) \cup LSD(\phi_D)\} \quad (5.1)$$

Such set is composed by norms whose components can be either more or less specific than the corresponding components of the original norm. We call a norm belonging to $\mathcal{A}(n)$ an *alteration* of the norm n. For example, an alteration of n may be a new norm $n' = (\phi'_C, P(\phi'_P), \phi'_D)$ such that ϕ'_C is less specific than ϕ_C, while ϕ'_P and ϕ'_D are more specific than ϕ_P and ϕ_D, respectively (i.e., such that $\phi'_C \in LSC(\phi_C)$, $\phi'_P \in MSP(\phi_P)$ and $\phi'_D \in MSD(\phi_D)$).

Alterations of a norm n prohibit a different set of behaviors than n. In the general case, the prohibited behaviors are neither a subset nor a superset of the behaviors prohibited by the original norm.

Different subsets of $\mathcal{A}(n)$, characterize different types of revisions of a norm. We consider two types of revisions: *weakening* and *strengthening*.

- *Weakening* a norm n generates n', which prohibits a *subset* of the behaviors prohibited by n. To weaken a norm, we can make the condition more specific, the prohibited state more specific, or the deadline less specific. The subset of $\mathcal{A}(n)$ that defines the set of all possible weaker versions of n is $\mathcal{W}(n)$, defined as per Eq. 5.2, where the sets $MSx(\phi_x)$ and $LSx(\phi_x)$, for $x \in \{C, P, D\}$, are as per above. Formally,

$$\mathcal{W}(n) = \{n' \mid n' = (\phi'_C, P(\phi'_P), \phi'_D) \;\&\; \phi'_C \in MSC(\phi_C) \;\&\; \phi'_P \in MSP(\phi_P) \;\&\; \phi'_D \in LSD(\phi_D)\} \quad (5.2)$$

- *Strengthening* a norm n generates n', which prohibits a *superset* of the behaviors prohibited by n. To strengthen a norm, we can make the condition less specific, the prohibited state less specific, or the deadline more specific. The subset of $\mathcal{A}(n)$ that defines the set of all possible strengthening of n is $\mathcal{S}(n)$, defined as per Eq. 5.3, where the sets $MSx(\phi_x)$ and $LSx(\phi_x)$, for $x \in \{C, P, D\}$, are as per

Table 5.3: *Summary of the heuristic revisions of the different components of a conditional norm.*

		Condition	State	Deadline
Prohibition	Make more specific	Make ϕ_C more specific Take CS, set of states s from violating traces s.t. ϕ_C holds. Let Φ = set of possible conjunctions from $(prop(CS) \cap L_C) \backslash prop(\phi_C)$ $MSC(\phi_C) = \{\phi_C \wedge \phi \mid \phi \in \Phi\}$	Make ϕ_P more specific Take PS, set of states s from violating traces s.t. ϕ_P holds. Let Φ = set of possible conjunctions from $(prop(PS) \cap L_P) \backslash prop(\phi\_P)$ $MSP(\phi_P) = \{\phi_P \wedge \phi \mid \phi \in \Phi\}$	Make ϕ_D more specific Take DS, set of states from obeying traces s.t. ϕ_D hold Let Φ = set of possible conjunctions of prop from $(prop(DS) \cap L_D) \backslash prop(\phi_D)$ $MSD(\phi_D) = \{\phi_D \wedge \phi \mid \phi \in \Phi\}$
	Make less specific	Make ϕ_C less specific Take OPS, set of states s from obeying traces s.t. ϕ_P holds. Let Φ = set of possible conjunctions from $prop(OPS) \cap L_C$ $LSC(\phi_C) = \{\phi_C \vee \phi \mid \phi \in \Phi\}$	Make ϕ_P less specific Take IPS, set of states s from obeying traces between states where ϕ_C and ϕ_D hold Let Φ = set of possible conjunctions from $(prop(IPS) \cap L_P)$ $LSP(\phi_P) = \{\phi_P \vee \phi \mid \phi \in \Phi\}$	Make ϕ_D less specific Take CPS, set of states s from violating traces between states where ϕ_C and ϕ_P hold. Let Φ = set of possible conjunctions from $prop(CPS) \cap L_D$ $LSD(\phi_D) = \{\phi_D \vee \phi \mid \phi \in \Phi\}$
Obligation	Make more specific	Make ϕ_C more specific Take CS, set of states s from violating traces s.t. ϕ_C holds. Let Φ = set of possible conjunctions from $(prop(CS) \cap L_C) \backslash prop(\phi_C)$ $MSC(\phi_C) = \{\phi_C \wedge \phi \mid \phi \in \Phi\}$	Make ϕ_O more specific Take OS, set of states s from obeying traces s.t. ϕ_O holds. Let Φ = set of possible conjunctions from $(prop(OS) \cap L_O) \backslash prop(\phi\_O)$ $MSO(\phi_O) = \{\phi_O \wedge \phi \mid \phi \in \Phi\}$	Make ϕ_D more specific Take DS, set of states from violating traces s.t. ϕ_D hold Let Φ = set of possible conjunctions of prop from $(prop(DS) \cap L_D) \backslash prop(\phi_D)$ $MSD(\phi_D) = \{\phi_D \wedge \phi \mid \phi \in \Phi\}$
	Make less specific	Make ϕ_C less specific Take ODS, set of states s from obeying traces s.t. ϕ_D holds. Let Φ = set of possible conjunctions from $prop(ODS) \cap L_C$ $LSC(\phi_C) = \{\phi_C \vee \phi \mid \phi \in \Phi\}$	Make ϕ_P less specific Take IOS, set of states s from violating traces between states where ϕ_C and ϕ_D hold Let Φ = set of possible conjunctions from $prop(IOS) \cap L_O$ $LSP(\phi_P) = \{\phi_P \vee \phi \mid \phi \in \Phi\}$	Make ϕ_D less specific Take COS, set of states s from obeying traces between states where ϕ_C and ϕ_O hold. Let Φ = set of possible conjunctions from $prop(COS) \cap L_D$ $LSD(\phi_D) = \{\phi_D \vee \phi \mid \phi \in \Phi\}$

above. Formally,

$$\mathcal{S}(n) = \{n' \mid n' = (\phi'_C, P(\phi'_P), \phi'_D) \;\&\; \phi'_C \in LSC(\phi_C) \;\&\; \phi'_P \in LSP(\phi_P) \;\&\; \phi'_D \in MSD(\phi_D)\} \quad (5.3)$$

Note that some of the combinations of revised components in the norms in the set (and subsets of) $\mathcal{A}(n)$ may not be relevant in the domain in which they need to be used. For example, in our highway scenario, we do not consider, nor intend to regulate, the behavior of the agents outside the highway section. Consider a norm $n' = (km_6, P(sp_{36}), km_4)$, which is detached at the 6-th km of the highway section, and prohibits speeds higher than 36 km/h until the 4-th km of the highway is reached. In our scenario, all vehicles that reach the 10-th km of the highway exit the simulation; hence, the finite traces describing their behaviors will never contain a state where km_4 holds after the state where the norm is detached. This norm in this scenario is therefore not relevant (even though valid, as practically analogous to a norm $n'' = (km_6, P(sp_{36}), \bot)$). Domain-specific constraints, if any, could be applied here to remove, from the set of revised norms, the norms that are not relevant. This is not, however, a required step, since the norms synthesized with our revision operations are are all valid (also the ones that are not relevant in a certain domain) w.r.t. to the language defined for their components.

On how to decide how to revise a norm. In some cases, one may want to constrain the revision of the norms to a specific type, like the weakening and strengthening types, or other possible subsets of $\mathcal{A}(n)$. Based on the context of application, one may devise strategies to decide what type of revision to perform. As a simple example, one may decide to strengthen a norm in case the number of false positives is higher than the number of false negatives, and to weaken the norm otherwise. Some examples of strategies can be found in [121]. In our experiments in Sec. 5.6, we show results by separately performing both alteration, weakening, and strengthening of a norm, unconditionally (i.e., without applying any strategy to decide what type of revision to perform) and we focus on the evaluation of the revised norms and on their comparison with the original ones. In future work, we will integrate our revision in a runtime context and use some of the above mentioned strategies.

5.5.3 Choosing The New Norm, the *Selection Step*

The *synthesis step* provides a set of new revised norms, which we call $\mathcal{R}(n)$ (e.g., $\mathcal{R}(n) = \mathcal{A}(n)$, or $\mathcal{R}(n) = \mathcal{W}(n)$). $\mathcal{R}(n)$ defines the search space through which we shall search for a new norm. In this section, we discuss the *selection step* that choose the new norm from $\mathcal{R}(n)$.

Consider the confusion matrix reported in Fig. 5.1a, which describes the relationship between the classification of traces according to a norm (i.e., whether a trace obeys or violates the norm) and the correct classification of the traces according to the MAS objectives labeling. Each cell (i, j) in the matrix contains the number of traces in the dataset Γ that were classified as i by the MAS objectives and as j by

the norm. For example, cell $(true, ob)$ contains the number of traces in Γ obeying the norm and achieving the MAS objectives, i.e., the number of true positives (TP). The inner diagonal of the matrix (the diagonal from TP to TN), represents the number of traces correctly classified by a norm. The outer diagonal, instead, represents the number of errors, or misclassifications.

$$\begin{array}{cc} & \text{norm} \\ & \begin{array}{cc} ob & viol \end{array} \\ \text{objectives} \begin{array}{c} true \\ false \end{array} & \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \end{array} \qquad \begin{array}{cc} & \text{norm} \\ & \begin{array}{cc} ob & viol \end{array} \\ \text{objectives} \begin{array}{c} true \\ false \end{array} & \begin{bmatrix} x & 0 \\ 0 & |\Gamma| - x \end{bmatrix} \end{array}$$

(a) **(b)**

Figure 5.1: *Confusion Matrix (a) and an example of confusion matrix with no misclassifications (b).*

Given a dataset of traces Γ labeled according to some MAS objective, the confusion matrix provides a compact representation of how well a norm is aligned with the MAS objectives, as per Sec. 5.3.1. A perfectly aligned norm w.r.t. Γ corresponds to the confusion matrix in Fig. 5.1b, where all traces are correctly classified and no trace is misclassified. Less aligned norms, instead, correspond to a matrix where FP and FN cells are non-zero integers. By analysing a confusion matrix, we can determine how many errors a certain norm is making, what type of errors (i.e., whether negative traces are considered positive more often than positive traces are considered negative), and we can quantitatively determine whether a norm is better (aligned with the MAS objectives) than another.

Confusion matrices are typically used in machine learning to evaluate classifiers. In this work, we consider a norm as the model of a binary classifier that distinguishes positive from negative traces. The literature proposes several metrics that combine the elements in the confusion matrix in order to compare different classifiers by means of a single value. One of such metrics is *accuracy*, defined as $acc(c, \Gamma) = \frac{TP+TN}{|\Gamma|}$, with TP and TN number of true positives and true negatives obtained with a classifier c on the dataset of traces Γ. We can use accuracy to characterize the concept of *alignment* of a norm with the MAS objectives. Given the set of revised norms $\mathcal{R}(n)$, we can choose as a revision of n the norm with highest accuracy, i.e., $n' = argmax_{r \in \mathcal{R}(n)}\ acc(r, \Gamma)$.

As an example, consider the three confusion matrices in Table 5.4, which are determined by the norms in $\mathcal{R}(n) = \{n_1, n_2, n_3\}$ obtained with the heuristic above presented on a dataset Γ that consists of 12 traces, 6 of which are positive traces and 6 are negative. In the context of evaluating the alignment of a norm, we sometimes use its corresponding confusion matrix since the confusion matrix represents the correct and mistaken classifications of the norm. Norm n_1 classifies correctly 75% of the traces making few errors among the positive traces and slightly more errors among the negative traces. Norm n_2 classifies correctly only 50% of traces. n_2 is weaker than n_1 and, thus, classifies many negative traces as positive. Finally, n_3 classifies

Table 5.4: *Accuracy of three examples of norms.*

	n_1	n_2	n_3
$\begin{bmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{bmatrix}$	$\begin{bmatrix} 5 & 1 \\ 2 & 4 \end{bmatrix}$	$\begin{bmatrix} 4 & 2 \\ 4 & 2 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 \\ 0 & 6 \end{bmatrix}$
acc	0.75	0.5	0.66

correctly 66% of traces. The norm is stricter than the others, and it captures all negative traces. This, however, comes at the cost of misclassifying many positive traces. By using the accuracy to choose a norm in $\mathcal{R}(n)$ we select then n_1 since its accuracy is the highest, as reported in Table 5.4.

Metrics for Norm Alignment with MAS objectives

Consider the three confusion matrices in column *a* of Table 5.5, obtained on a dataset Γ composed by 12 traces, 6 of which are positive traces and 6 are negative. Suppose the three matrices represent the alternative norms we have to choose from:

- The first norm is a very weak norm: every trace in Γ obeys the norm. Such norm correctly classifies all the positive traces but misclassifies all negative traces.
- The second norm is a less weak norm: some traces in Γ violates the norm. Such norm misclassifies traces in both classes, making slightly more errors among the negative traces.
- The third norm is a more balanced norm: like all the other norms, it correctly classifies half of the traces in Γ, however the errors are uniformly distributed between positive and negative traces.

Such three norms are very different. Yet, the *accuracy* of all these norms is the same (i.e., 0.5), as they all correctly classify half of the dataset. Which one shall we select as a revision of the current norm?

Table 5.5: *Comparison of accuracy with other metrics on different examples of norms.*

	a			*b*	
$\begin{bmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{bmatrix}$	$\begin{bmatrix} 6 & 0 \\ 6 & 0 \end{bmatrix}$	$\begin{bmatrix} 4 & 2 \\ 4 & 2 \end{bmatrix}$	$\begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}$	$\begin{bmatrix} 99 & 0 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 98 & 1 \\ 0 & 1 \end{bmatrix}$
acc	0.5	0.5	0.5	0.99	0.99
F1	0.6667	0.5714	0.5	0.995	0.9949
*MCEN**	0.7143	0.1461	0.0943	0.9559	0.9493
Kappa	0	0	0	0	0.6622

Take, as a different example, the two norms in column b of Table 5.5. This time, the dataset Γ includes 100 traces, 99 of which are positive traces and only 1 is a negative trace. Such Γ is an example of imbalanced dataset, where the vast majority of traces belongs to one of the two classes. The first norm correctly classifies all the positive traces, but misclassifies all negative traces (even if it's only one). The second norm, instead, classifies correctly the only negative trace, and almost all the positive traces, misclassifying only one of them. Due to the imbalanced nature of the dataset, however, the *accuracy* of both norms is the same, as 99% of traces are correctly classified in both cases. Again, which norms shall we select?

Since in some contexts, false positives may be less important than false negatives, or vice-versa, a number of metrics have been proposed as alternatives to accuracy to assess the quality of a classifier. Some well-known metrics include the *F-measure (F1)* [266], the *Confusion Entropy* [291], the *Kappa coefficient* [89]. Table 5.6 reports their definition. Some of them deal better with imbalanced datasets. For example, *F1* is a measure that seeks a balance between precision and recall by considering the class of positives as more important. Cohen's Kappa is a measure which compares the observed accuracy with the expected accuracy that can be reached by random chance on the given dataset, and for this reason it is typically used on problems involving imbalanced data. Other metrics are based on different principles. The Modified Confusion Entropy *mcen* [116], for example, is a metric based on the concept of Shannon's Entropy [251], that measures the entropy generated from misclassified cases. In doing so it considers both the cases where each class is misclassified into other classes, and the cases where the other classes are misclassified as belonging to the considered class. Generally speaking, higher accuracy tends to result in lower Confusion Entropy. In order to ease the comparison, in the following we use the inverse of *mcen* (i.e., $mcen^* = (1 - mcen)$) so that we have higher values for norms that are considered better and lower values for norms that are considered worse.

Table 5.6: *Examples of metrics for the evaluation of a binary classifier. S indicates the size of the dataset Γ. acc is the accuracy; F1 is the F-measure, where Precision is defined as $\frac{TP}{TP+FP}$, and Recall as $\frac{TP}{TP+FN}$; mcen is the Modified Confusion Entropy; kappa is the Cohen's Kappa, where $expAcc = \frac{((TN+FP)(TN+FN))+((FN+TP)(FP+TP))}{S \cdot S}$.*

Metric	Definition	Scale
acc	$\frac{TP+TN}{S}$	$[0,1]$
F1	$2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$	$[0,1]$
mcen	$\frac{2(FN+FP)log((S-TN)(S-TP))}{3 \cdot S+FN+FP} - \frac{4((FN \cdot log(FN))+(FP \cdot log(FP)))}{3 \cdot S+FN+FP}$	$[0,1]$
kappa	$\frac{acc-expAcc}{1-expAcc}$	$[-1,1]$

Compare these metrics with accuracy in Table 5.5. Unlike accuracy, *F1* distinguishes all norms in the Table. According to *F1*, in both columns a and b of Table 5.5 the first norm is the best one. This occurs because more positive traces, the prioritized class by *F1*, are correctly classified in both cases. The rankings of the norms given by

*mcen** is analogous to the ranking of *F1* in the reported examples. Notice, however, how in column *a* of Table 5.5, the first norm is considered significantly better than the others. This happens because the traces are distributed only in two groups, thereby exhibiting lower entropy. Finally, notice how *kappa* considers the second norm in column *b* of Table 5.5 significantly better than the first one[2]. This is because for the first norm the expected accuracy *expAcc* is already 0.99, so reaching an accuracy of 0.99 corresponds to reaching the same agreement that could be obtained if both the norm and the MAS objectives classified the traces as positive randomly (for this reason *kappa* gets its middle value). For the second norm, the expected accuracy is instead around 0.97, so reaching an accuracy of 0.99 provides a significant improvement over the expected random agreement.

Without going into the details of each specific metric, beyond the scope of this paper and extensively studied in the above mentioned works, we emphasize that that we propose supports the use of any of the above described (or other) metrics to assess the alignment of a norm in the *selection step*. In different contexts, therefore, *DNR* can be tuned with the most opportune metric.

5.5.4 Multiple Norms

Revising a set of norms $\mathcal{N}$ generates a set of norms $\mathcal{N}'$ such that at least one of the norms in $\mathcal{N}$ is revised. A set of norms $\mathcal{N}$ is weakened (strengthened) when only weakening (strengthened) is applied, otherwise the set is altered. In Sec. 5.5.3, we interpreted a norm as a binary classifier. We can generalize this concept to a set of norms. We interpret the set of norms as a multi-label binary classifier, where multiple binary labels can be assigned to each trace. Each norm assigns a different binary label to a trace.

Consider, as an example, a set $\mathcal{N} = \{n_1, n_2\}$ composed by two norms and a dataset of traces Γ. A trace $t \in \Gamma$ can obey n_1 and violate n_2, thereby two labels $n_1 ob$ and $n_2 viol$ are assigned to the trace t.

$$
\begin{array}{cc}
 & \begin{array}{cccc} & n_1\ ob & & n_1\ viol \\ n_2\ ob & n_2\ viol & n_2\ ob & n_2\ viol \end{array} \\
\text{objectives} \begin{array}{c} true \\ false \end{array} & \begin{bmatrix} PFC & PPC_1 & PPC_2 & PFW \\ NFW & NPC_2 & NPC_1 & NFC \end{bmatrix}
\end{array}
$$

Figure 5.2: *A confusion matrix for the case of two norms n_1 and n_2. PFC stands for Positive Fully Correct (i.e., number of positive traces in a dataset Γ correctly classified by both norms); PPC_i stands for Positive Partly Correct, with i indicating the id of the norm that classifies correctly; PFW for Positive Fully Wrong. NFW, NPC_i and NFC are analogous for the Negative traces.*

[2]Note that the values of *kappa* range from -1 to 1, so value 0 is the middle value for the metric.

Unlike the case of a single norm, where a trace could be either correctly or wrongly classified, a trace can now be also partly correctly classified. This happens, for example, if a trace is a positive trace and it obeys one norm but violates another norm. Fig. 5.2 reports a representation of a confusion matrix for the case of two norms, that illustrates this aspect. Suppose we aim to revise $\mathcal{N}$ by strengthening n_1 and weakening n_2. One straightforward way to apply our approach is the following: order the norms in some way and revise them one by one, performing independently for each of them the *synthesis* and *selection steps*.

Revising each norm independently may lead to a set of new norms that, while each is better aligned with the MAS objectives, their combination diminishes the number of fully correct classifications compared to the original set. Fig. 5.3 shows an example where a set of norms $\mathcal{N} = \{n_1, n_2\}$ is revised by weakening n_1 and strengthening n_2, obtaining a new set $\mathcal{N}'$. While the number of correctly classified traces by each norm increases after the revision, the number of traces correctly classified by both of them at the same time decreases after the revision, increasing instead the number of partly correct classifications. Comparing the values inside and outside the brackets in Fig. 5.3b, we notice that after the revision all traces are only partly correctly classified.

	$\mathcal{N}$			$\mathcal{N}'$	
Trace	n_1	n_2	Objectives	n_1	n_2
ρ_1	viol	viol	true	ob	viol
ρ_2	ob	ob	false	ob	viol
ρ_3	viol	ob	true	ob	viol
ρ_4	viol	viol	false	ob	viol

(a)

objectives	n_1 *ob*, n_2 *ob*	n_1 *ob*, n_2 *viol*	n_1 *viol*, n_2 *ob*	n_1 *viol*, n_2 *viol*
true	0(0)	0(2)	1(0)	1(0)
false	1(0)	0(2)	0(0)	1(0)

(b)

Figure 5.3: *Example of classification of four traces by a set of norms $\mathcal{N}$ and by its revision $\mathcal{N}'$ (a) and the corresponding confusion matrix (b). The values in the matrix in between brackets refer to the set $\mathcal{N}'$.*

If having fully correctly classified traces is not important, for example if it is sufficient for the MAS objectives to being achieved that one, or some, of the norms is obeyed, then revising each of them individually may be a good strategy and the alignment of the norm set can be determined as the average alignment of all the norms. If, conversely, it is important to have fully correctly classified traces, instead of revising each norm independently, we can search for a combination of norms that minimizes the combined errors. Similarly to the case of one norm, we can look for the *set* of norms that is most aligned with the MAS objectives. This time, however, the alignment of the norm set must be determined w.r.t. the whole set of norms at once. As per the single label case, the literature offers several metrics that can be used to capture this notion for a multi-label classifiers [259].

A direct generalization of the accuracy to multi-label problems, sometimes called Jaccard index [224], for example, is defined as the average across all traces of the proportion of the predicted correct labels to the total number (predicted and actual) of labels for each trace. Eq. 5.4 reports its formalization. Z_i is the list of labels predicted by the norms in $\mathcal{N}$ for a trace i, and Y_i is the list of correct labels for trace

i according to the MAS objectives.

$$ml\text{-}acc = \frac{1}{|\Gamma|}\sum_{i=1}^{|\Gamma|}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{5.4}$$

Other well-known metrics for assessing the quality of a multi-label classifier include the Hamming Loss [244] or the Weighted Kappa [90]. We do not go into their details, but we observe that, as per the case of a single norm, all these metrics (and other) can be used in *DNR* to assess the alignment of a norm set with the MAS objectives.

5.5.5 On the Complexity of the Heuristic Revision

The complexity of the approach presented in this section depends on two factors: the complexity of determining the alignment of a norm (set) with the MAS objectives, and the size of the set of norms that are compared in the *argmax* operator. Using a measure such as accuracy to calculate the norm alignment has a complexity that is linear in the size of Γ. The size of the set of norms to be compared depends on the number of components of a norm that are revised (in the worst case, 3: both condition, prohibited state and deadline) and on the size of the set Φ of combinations of propositions considered for the revisions of a component. If we bound the maximum size of Φ to a polynomial in the number of propositions, the resulting complexity of generating each set of revised components is polynomial in $|\Gamma|$. In the worst case, an operation of strengthening or weakening, therefore, generates a number of new norms to be compared bounded by a cubic polynomial. Finally, the number of comparisons grows exponentially with the number of norms that cannot be revised independently. This is because for each norm being revised, a set of possible new norms is generated. The new possible norm sets to be compared, therefore, are obtained by combining the norms from all such sets. Note, however, that if the number of norms is high, different strategies could be adopted to bound the number of norm sets to compare, analogously to the bounding of the set Φ.

5.6 Experimentation

We report on experimental results concerning *DNR*, presented in Sec. 5.5. Our experiments aim to provide an empirical answer to the following questions:

Q1. *To what extent is the synthesis step leading to norms that reduce the number of classification errors?*

Q2. *How does the choice of a metric for norm-alignment affect the revision?*

Q3. *How does DNR affect the alignment between norms and MAS objectives?*

Q4. *How does DNR generalize to previously unseen traces?*

To answer **Q1-4**, we make use of a traffic simulation of the highway scenario of Sec. 5.3 implemented with the SUMO traffic simulator [184] and we set up our experiments as follows. We consider a single conditional prohibition n, which regulates

the maximum speed of the vehicles in the highways section. In particular, we focus on a norm whose components consist of a single disjunct (i.e., the norm only has one sub-norm). In Sec. 5.6.1, we will then report also on additional experimental results explicitly concerning multiple norms and the use of multi-label metrics. Moreover, we consider the two agent types (*car* and *truck*) described in Sec. 5.3. Both agent types aim at driving through the highway section by maximizing their speed. At each simulation step, every agent determines its desired speed according to its internals and to the currently enforced norms. Every agent also exhibits a number of behaviors which are not regulated by our norm and are therefore not monitored, such as overtaking the vehicle ahead, signaling the overtaking, accelerating or braking, etc. We use a population of agents (arbitrarily) uniformly distributed among *cars* and *trucks*, where 75% of the agents is always compliant with the enforced norms, while the remaining 25% will ignore them and focus on maximizing speed. This can be seen as enforcing the norms by means of sanctions that can be afforded by 25% of agents. Note that, while compliant agents always obey the norms, agents that ignore the norm do not necessary violate it, as their ability to violate the speed limit depends both on their type (i.e., the vehicle's maximum speed) and on the surrounding environment (e.g., traffic jams will force agents to slow down, regardless of their preferences). Throughout the simulations, we collect execution traces that describe the behavior of each agent. An execution trace of an agent is composed of 10 different states, one per each of the 10 kilometers of the highway section. The i-th state of a trace contains information about: (i) the i-th km of the highway, (ii) the maximum speed the agent reached in the i-th km of the highway, (iii) the type of agent. An example trace is the following, describing the behavior of a car in the highway.

$$\begin{aligned}(\{km_1, sp_{30}, car\}, \{km_2, sp_{22}, car\}, \{km_3, sp_7, car\}, \{km_4, sp_{32}, car\},\\ \{km_5, sp_7, car\}, \{km_6, sp_{18}, car\}, \{km_7, sp_{32}, car\},\\ \{km_8, sp_7, car\}, \{km_9, sp_{18}, car\}, \{km_{10}, sp_{14}, car\})\end{aligned} \quad (5.5)$$

Each trace is labeled w.r.t. the CO_2 emitted by the vehicle on the highway and the time needed to travel from the beginning to the end of the highway section. A trace is labeled as *positive* if the maximum emission of the vehicle from the beginning to the end of the highway section was below a threshold t_{co2} (in our experiments, we used $t_{co2} = 100$ g/s [3]) and the travel time was below a threshold t_{tt} (in our experiments $t_{tt} = 450s$, the time it would take to drive for 10 km at 80 km/h), and *negative* otherwise. We emphasise once more that *DNR* is agnostic of such underlying rules for the labeling and it is exposed only to the given labeled traces.

In the following, when we report statistics, we use M, SD, SE, Min, Q1-Q3, and Max to indicate mean, standard deviation, standard error, minimum value, the three quartiles, and the maximum value, respectively.

[3] As a reference, the default truck model in the SUMO traffic simulator can reach up to 120 g/s of CO_2.

Q1. To what extent is the *synthesis step* leading to norms that reduce the number of classification errors?

We study the composition of the sets of norms generated during the *synthesis step*, as per Sec. 5.5.2. We expect the set of altered norms $\mathcal{A}(n)$, to be composed by norms with higher number of TP and TN traces compared to the original norms, and that lower number of FP and FN. Perfect alterations would neither reduce the number of TP and TN, nor increase the number of FP and FN. An approximate approach like ours, however, will likely exhibit a certain amount of undesired *side effects* (reducing in some cases the TP and TN, and increasing the FP and FN). The better the heuristic, the fewer side effects. With set $\mathcal{W}(n)$, we expect an higher number of obeying traces, hopefully only TP, and a lower number of violating traces, hopefully only FN. Finally, with set $\mathcal{S}(n)$ we expect an increase in the violating traces, hopefully only TN, and a decrease in the obeying traces, hopefully only FP.

In order to answer the question, we adopt the following method.

1. We execute a simulation of the highway scenario where a norm n, randomly selected from the set of all possible norms, is enforced.

2. The simulation runs until 1500 vehicles drive through the highway section. Since the behavior of each vehicle corresponds to an execution trace, the simulation generates a dataset Γ containing 1500 traces. The traces are labeled, as per Table 5.1, w.r.t. the obedience of norm n and the achievement of the MAS objectives above described.

3. Given Γ, the *synthesis step* is performed to generate a set of revised norms $\mathcal{R}(n)$.

4. The confusion matrices of the synthesized norms are compared with the confusion matrix of the original norm n w.r.t. the *same dataset*[4] Γ. To obtain a reliable and statistically significant comparison, we repeat 100 times the steps 1-3, so that each time the simulation is run, a different norm is enforced. We execute therefore 100 independent simulations, in each of which we enforce a different norm so to obtain 100 different and independent dataset Γ_{1-100}, which we use to generate 100 different *sets* of revised norms. The results that we report refer to such 100 repeated trials.

5. We repeat steps 3-4 for the three types of revision as per Sec. 5.5.2, synthesizing in step 3 the sets $\mathcal{R}(n) = \mathcal{W}(n)$, $\mathcal{R}(n) = \mathcal{S}(n)$ and $\mathcal{R}(n) = \mathcal{A}(n)$, respectively for weakening, strengthening and alteration.

Table 5.7 reports an overview, w.r.t. datasets Γ_{1-100}, of the 100 norms that we revise in our experiments. On average, the datasets are split relatively evenly among positive and negative traces, while, on average, the violating traces are about the 7% of the total. Note also that the enforced norms were generally too weak: while they covered relatively well the positive traces (high values of TP), in many cases they mis-classified a large part of the negative traces (high values of FP). This is reflected

[4]We emphasize again, for clarity, that in order to compare the original norm with the new norms we study their confusion matrices w.r.t. the same set of traces Γ. We do *not* enforce in the system the revised norms, as we focus here on the realignment of a norm with the MAS objectives w.r.t. Γ.

Table 5.7: *Statistics about the 100 original norms enforced in our simulations w.r.t. datasets* Γ_{1-100}. *Values for one norm* n_i *are calculated w.r.t.* Γ_i, *composed by 1500 traces.*

	TP	FP	TN	FN	precision	recall	*acc*	*F1*	*mcen**	*kappa*
M	0.485	0.443	0.043	0.028	0.511	0.878	0.528	0.602	0.623	-0.003
SD	0.271	0.234	0.063	0.0372	0.287	0.237	0.219	0.318	0.186	0.081
Min	0	0	0	0	0	0	0.100	0	0.280	-0.170
Q1	0.315	0.323	0	0	0.364	0.875	0.356	0.495	0.453	-0.009
Q2	0.621	0.374	0.0003	0.001	0.626	0.950	0.626	0.768	0.685	0
Q3	0.6262	0.5730	0.0558	0.0616	0.6493	1	0.626	0.770	0.687	0
Max	0.92	0.90	0.25	0.14	1	1	0.930	0.960	0.930	0.280

also in the relatively low value of the precision of the norms. Finally, the average accuracy is about 53%, with a maximum value of 93% and a value below 63% for 75% of the norms (see the third quartile Q3).

Table 5.8 reports, instead, an overview of the norms in the 100 different sets generated by the *synthesis step* for the three types of revisions. In the following, we study how the norms in these sets differ from the original norms described in Table 5.7.

Fig. 5.4 reports three box plots illustrating the changes in the percentage of TP, FP, TN, FN (e.g., for TP, the difference between the percentage of TP with a revised norm and with the original norm). The box plots describe the changes for the norms synthesized in the 100 repeated trials. Fig. 5.4d reports the detailed statistics of the changes, together with the effect size d_{Cohen}. The effect size is a statistical measure that describes the strength of a phenomenon, by computing the difference between two groups of measurements in terms of their common standard deviation [87]. We use d_{Cohen} to analyze the change in the percentage of TP, FP, TN and FN, and to understand if the effect of such change has a statistically relevant magnitude.

In the case of weakening (Fig. 5.4a), we observe, as desired, a *large*[5] reduction of the number of FN traces. Notice how the reduction of FN corresponds to an increase of TP, showing that those traces are now correctly permitted. The revision also exhibits a large *side effect* of reducing TN with a corresponding increase of FP. We do not see any negative change for the positive traces, nor any positive change for the negative traces. This is not surprising and simply validates the correctness of the heuristic operations: a weakening operation does not affect negatively the positive traces, and a strengthening operation does not affect negatively the negative traces.

Analogously, in the case of strengthening (Fig. 5.4b), we see, as desired, a large decrease in the number of FP traces. Also in this case the revision exhibits a large side effect of reducing the number of TP traces. Note how the changes are bigger for strengthening than for weakening. This, again, is because in the considered datasets there are no more than 25% violating traces, while up to 100% obeying traces.

Since we performed weakening or strengthening revisions regardless of the actual number of FP or FN traces, in some cases no revision was needed or possible (e.g., when FP or FN were 0). In our simulations, this was more common for weakening

[5]Please note that we use the terms *small*, *intermediate* and *large* according to the interpretation of the effect size d_{Cohen} [87]. A change is considered having *no effect* if $|d_{Cohen}| < 0.2$; *small effect*, if $0.2 \leqslant |d_{Cohen}| < 0.5$; *intermediate effect*, if $0.5 \leqslant |d_{Cohen}| < 0.8$; and *large*, if $|d_{Cohen}| \geqslant 0.8$.

Table 5.8: *Statistics about the size (a) and the 100 sets of synthesized norms (b).*

(a)

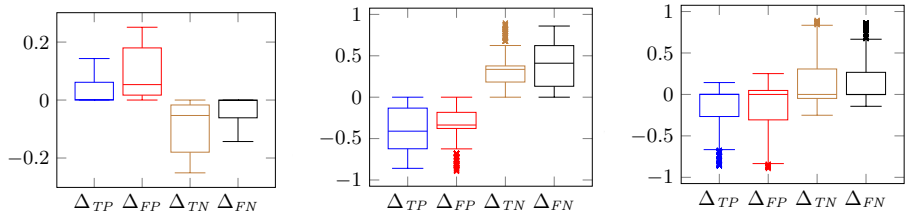
Type of Revision	Set	M	SD	SE	Tot. norms
Weakening	$\mathcal{W}(n)$	35.67	67.87	6.79	3567
Strengthening	$\mathcal{S}(n)$	14.81	14.19	1.42	1481
Alteration	$\mathcal{A}(n)$	96.11	117.20	11.72	9611

(b)

		TP	FP	TN	FN	precision	recall	*acc*	*F1*	*mcen**	*kappa*
$\mathcal{W}(n)$	M	0.311	0.95	0.357	0.46	0.316	0.738	0.323	0.374	0.823	0.003
	SD	0.341	0.123	0.385	0.36	0.347	0.41	0.335	0.392	0.174	0.066
	Min	0	0.01	0	0	0	0	0	0	0.3	-0.15
	Q1	0.001	0.959	0	0	0.001	0.667	0.005	0.001	0.675	-0.005
	Q2	0.066	0.993	0.222	0.511	0.067	0.989	0.129	0.125	0.917	-0.001
	Q3	0.636	0.998	0.75	0.8	0.646	1	0.643	0.778	0.984	0
	Max	0.99	1	1	1	1	1	1	1	1	0.74
$\mathcal{S}(n)$	M	0.153	0.077	0.466	0.957	0.855	0.251	0.515	0.257	0.667	0.108
	SD	0.219	0.181	0.267	0.157	0.282	0.337	0.244	0.355	0.217	0.272
	Min	0	0	0	0	0	0	0.01	0	0.02	-0.75
	Q1	0	0	0.302	0.994	0.853	0	0.348	0	0.632	0
	Q2	0	0	0.374	0.999	1	0	0.433	0	0.736	0
	Q3	0.429	0.016	0.739	1	1	0.562	0.707	0.613	0.783	0.007
	Max	0.92	1	1	1	1	1	1	1	1	0.77
$\mathcal{A}(n)$	M	0.249	0.531	0.434	0.729	0.515	0.528	0.44	0.342	0.671	0.018
	SD	0.29	0.424	0.36	0.354	0.4	0.42	0.297	0.364	0.27	0.223
	Min	0	0	0	0	0	0	0	0	0.02	-0.75
	Q1	0	0.008	0	0.52	0.005	0	0.171	0	0.56	-0.008
	Q2	0.071	0.514	0.368	0.954	0.622	0.628	0.445	0.143	0.692	0
	Q3	0.502	0.987	0.755	0.999	0.964	0.984	0.655	0.755	0.91	0
	Max	0.99	1	1	1	1	1	1	1	1	0.77

operations (see the median values in Fig. 5.4a closer to 0). This happened, for instance, when we enforced a very weak speed limit so that all exhibited behaviors were permitted, or when, by enforcing a very strict speed limit norm, traffic jams were generated by compliant agents that significantly slowed down, preventing also any other agent behind them to violate the norm. Similarly, if the number of FP traces was 0, no strengthening was possible/needed. This happened, for instance, when the enforced speed limit was already relatively aligned with the MAS objectives, and it was strict enough not to allow vehicles to speed too much, but not too strict to cause jams as above described. When no revision was needed or possible, we did not revise the original norm, resulting in no change in the classification of traces.

When performing an alteration (Fig. 5.4c), we are not limited anymore to revisions concerning exclusively positive or negative traces, and the revision can affect all types of traces. We can see this in Fig. 5.4c by noting both an increase and a decrease in all types of traces. The sets of revised norms are more similar to the ones obtained with strengthening (at least in terms of change in the values of the confusion matrix). This shows that, in general, stricter norms were generated; this is in line with the fact



(a) **(b)** **(c)**

		M	SD	d_{Cohen}	Interpretation
$\mathcal{W}(n)$	Δ_{TP}	0.0288	0.03742	0.088	no effect
	Δ_{FP}	0.0989	0.08900	0.322	small
	Δ_{TN}	-0.0989	0.08900	-1.450	large
	Δ_{FN}	-0.0288	0.03742	-0.938	large
$\mathcal{S}(n)$	Δ_{TP}	-0.3897	0.25330	-1.750	large
	Δ_{FP}	-0.3291	0.19685	-1.948	large
	Δ_{TN}	0.3291	0.19685	2.027	large
	Δ_{FN}	0.3897	0.25330	2.030	large
$\mathcal{A}(n)$	Δ_{TP}	-0.1395	0.23643	-0.473	small
	Δ_{FP}	-0.1146	0.25589	-0.357	small
	Δ_{TN}	0.1146	0.25589	0.617	intermediate
	Δ_{FN}	0.1395	0.23643	0.780	intermediate

(d)

Figure 5.4: *The change of the % of TP, FP, TN, FN in the sets of norms $\mathcal{W}(n)$, $\mathcal{S}(n)$, $\mathcal{A}(n)$ respectively obtained by (a) weakening, (b) strengthening or (c) altering a norm, with their detailed statistics (d). Average values are obtained w.r.t. the 100 different sets of norms obtained revising the 100 original norms in the highway scenario. The percentage of change for each norm is calculated w.r.t. the total number of traces (i.e., 1500). d_{Cohen} and its interpretation refer to the effect size as per [87].*

that the original norms, as above described, were generally too weak. For this reason, the revision mostly affected the traces that were wrongly permitted: the FPs in the revised norms were reduced, even though with a small effect size. In some cases, the revisions also had the undesired side effect of increasing the number of FN traces. However, such effect was more marginal than with strengthening, indicating that when a dataset of traces includes both FP and FN traces, alteration operations permit to generate norms that are better aligned with the MAS objectives. In particular, the size of the effect on the FN traces had a magnitude more than 2 times smaller than that of the analogous effect for strengthening.

We conclude that the sets of synthesized norms are in line with the expectations laid down earlier in the section. The results show that, among the generated norms, some can have very significant side effects. This motivates us to propose the use of a metric to select among the possible norms. In the following, we show that using an appropriate metric is crucial to minimize the side effects of the revision.

Q2. How does the choice of a metric for norm-alignment affect the revision?

Using one metric or another in the *selection step* affects the precision and the recall of the resulting norms. We compare the precision and recall on the set of traces obtained when using different metrics, and discuss the implications in terms of normative concepts. We take the sets of norms $\mathcal{W}(n)$ $\mathcal{S}(n)$, $\mathcal{A}(n)$ generated for **Q1**, and study the norms that are selected when the metrics are used in the *selection step*. In particular, we analyze how the number of TP, TN, FP, and FN traces change w.r.t. the original norms. We compare the metrics also against a baseline unbiased metric, which we call *random*, that selects the new norm by randomly and uniformly sampling the sets $\mathcal{W}(n)$, $\mathcal{S}(n)$, $\mathcal{A}(n)$.

Fig. 5.5 illustrates such comparison via box plots. The reported values show the change of percentage of TP, TN, FP and FN traces with the 100 revised norms, obtained in the 100 trials, w.r.t. the original norm. Table D.1 reports the detailed statistics of the change illustrated in Fig. 5.5. In Tables D.2–D.4, instead, we provide the detailed statistics about the selected norms, respectively for weakening, strengthening and alteration. We include these tables in Appendix D for the sake of readability.

Consider the metric *acc* (second group of boxes in the plots). Compared to *random* (first group of boxes in the plots), the side effects of the revision are significantly reduced both for weakening and strengthening, and especially for alteration. For weakening, while with *random* the percentage of TN traces decreased, on average, of about 3.6%, with *acc* the reduction was significantly lower (about 0.8%, on average)[6]. Additionally, *acc* also provided slightly, but not significantly, better norms in terms of TP (the gained TP traces were about 2.5%, while with *random* they were 2.2%). For alteration, *acc* significantly improved in all classes compared to *random*, selecting norms with significantly (and with large effect size: $d_{Cohen} = 0.812$) more TN traces (and consequently less FP traces), and with a side effect on FN and TP which had a size about 1.5 times smaller than the effect with *random* (*acc*: $d_{Cohen} = 0.571$, *random*: $d_{Cohen} = 0.812$).

If we look at the changes in Fig. 5.5, and at Tables D.2–D.4, which report the average values of precision and recall of the selected norms, for weakening, strengthening and alteration, respectively, we can observe some differences between the different metrics.

Weakening. We aim at finding a new norm that is weaker than the original one, i.e., we aim at reducing the number of FN traces, hopefully without increasing too much the number of FP traces. In other words, we look for a new norm that has higher recall, without decreasing precision too much. We observe that:

- *random*, shows that the synthesized norms have, as desired, on average a higher recall than the average recall of the original norms (compare the recall of the new norms with the recall in Table 5.7), and a precision similar to the original norms.

[6]By conducting an independent-samples t-test to compare the change of the percentage TN traces with *random* (M=-0.0359, SD=0.05741) and with *acc* (M=-0.0077, SD=0.01514), we identify a significant difference; t(198)=4.760, p=0.000, with an intermediate effect size $d_{Cohen} = 0.672$

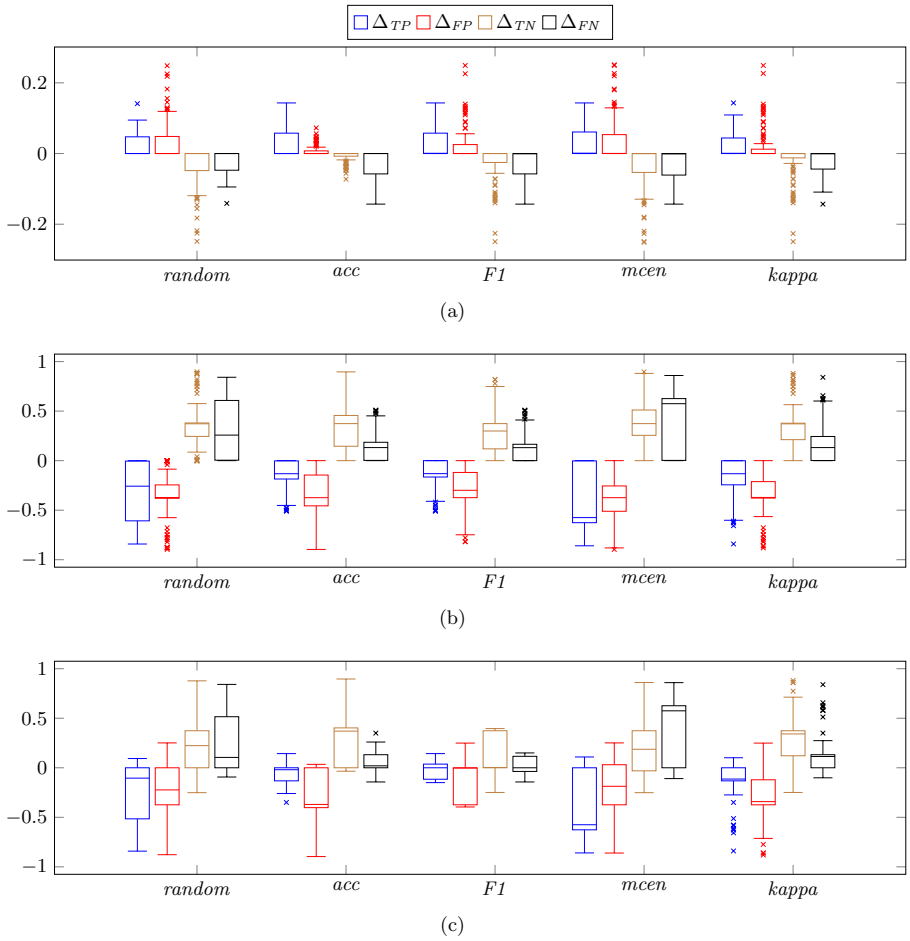


Figure 5.5: *Comparison of the change of the % of TP, FP, TN and FN after (a) weakening, (b) strengthening or (c) altering a norm with our heuristic when selecting a new norm according to different metrics.*

- *acc* selected norms that, in addition to having an high recall, about 92%, were also more accurate than the original norms, showing that the average sum of TP and TN increased as a whole compared to the original norms, hence the gain of new TP traces was higher than the loss of TN traces.
- *F1* selected norms that were weaker than *acc* (notice the higher recall around 98%). Note in Fig. 5.5a and Table D.1 how the decrease for the FN for *acc* and *F1* is similar, but the the increase for FP with *F1* is higher.
- *mcen** selected very weak norms. While providing a change for FN similar to *acc* and *F1*, *mcen** exhibited a more significant increase for FP, especially if compared to *acc*. This is because, as mentioned in Sec. 5.5.3, *mcen** tends to prefer norms that group traces in fewer classes, thereby reducing the entropy in the confusion matrix. Very weak norms (e.g., a norms whose condition is never detached), are such that a bigger number of traces is considered compliant and so traces are grouped in fewer classes. For this reason, we can see in Table D.2 that, among the tested metrics, *mcen** shows the highest recall and the lowest precision.
- *kappa* was the most balanced metric. On average, the selected norms had higher precision, compared to the other metrics, and we can see a lower decrease for FN.

Strengthening. We aim at finding a new norm that is stricter than the original one, i.e., we aim at reducing the number of FP traces, hopefully without increasing too much the number of FN traces. In other words, we look for a new norm that has higher precision, without losing too much recall.

- *random* shows that, as desired, the synthesized norms have, on average, higher precision than the average precision of the original norms, but also a very low recall (about 30%).
- *acc* selected stricter norms than *random* (note the higher precision), which were also more accurate and had an higher recall.
- *F1* selected less strict norms, with a lower precision (about 68%), even though it affected less significantly the FN (the average recall is about 64%; the highest among the tested metrics).
- *mcen**, for the reason explained above, was again the most "extreme" metric, selecting very strict norms, with a precision of about 92%, and very low recall (about 16%).
- *kappa* was again the most balanced metric, with values of precision and recall in between the values obtained with *acc* and *F1*.

Alteration. When altering a norm, we aim at improving the general alignment of the original norm with the MAS objectives, i.e., at both reducing the number of FP traces and the number of FN traces. In other words, we look for a new norm that balances precision with recall. Consider Table D.4.

- *random* shows that, overall, the synthesized norms have higher precision than the original ones, and lower recall. This illustrates that the revision operations generated on average stricter norms. However, the alignment of the selected norms did not significantly improve compared to the original norms. If we compare the values with the values in Table 5.7 we see that both *acc* and *F1* decreased, *kappa* slightly increased, and only *mcen** increased.

- *acc* selected stricter norms than the original ones (note the higher precision and lower recall) which were better aligned with the MAS objectives, with an average accuracy of 82%, misclassifying on average, only about 13% of traces.

- *F1* selected norms that were less strict than the ones selected by *acc* (see the lower precision and higher recall), and, as desired, they had a better alignment (in terms of *F1*) with the MAS objectives than the original norms.

- *mcen** selected norms with high precision and low recall, which led to norms that had very low entropy and therefore higher values of *mcen** compared to the original norms.

- *kappa* selected norms balancing precision and recall. Also in this case the average value of *kappa* improved compared to the average *kappa* of the original norms.

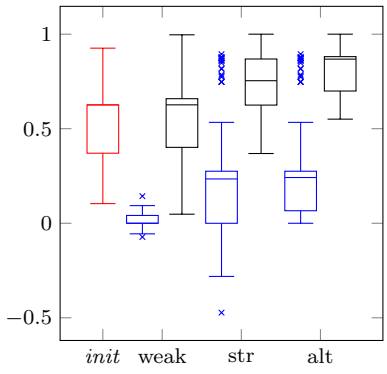
The results show that in all cases the alignment of the norms with the MAS objectives (defined by means of one of the metrics) improved after revising the norms. The norms selected by *mcen** were generally more "extreme", in the sense that they were either very weak or very strict, compared to the norms selected by the other metrics. This is because such norms tend to cluster traces in fewer categories, thereby reducing the entropy of the confusion matrix. This metric may be useful in cases where a radical change of the original norm in a specific direction is required (e.g., weakening a norm by completely disabling it). While *mcen** produced norms with significantly lower entropy, it did not select norms that balanced well precision and recall, and the selected norms did not always reduce the number of FP and FN traces. Both *acc*, *F1* and *kappa*, instead, provided less "extreme" revisions.

We illustrated some of the differences between the discussed metrics, which give different values to the precision and recall of a norm. In all cases, as we have seen comparing Fig. 5.5 with Fig. 5.4, the metrics provided a useful filter to select, from the set of possible norms generated in the *synthesis step*, the ones better aligned with the MAS objectives, discarding the ones with strong side effects.

Q3. How does *DNR* affect the alignment between norms and MAS objectives?

We now discuss the change in the alignment with the MAS objectives of the revised norms. We want to determine if the revised norms are better aligned with the MAS objectives. In **Q2** we already observed that in all cases the alignment with the MAS objectives improved when using the corresponding metric to select from the set of synthesized norms. We focus on the case where the the two classes of positive and negative traces are equally important, and we are concerned only with the absolute number of correctly classified traces. For this reason, we consider only metric *acc* to

select the new norm among the possible ones, and we use the norms selected by *acc* in **Q2** from the sets $\mathcal{W}(n), \mathcal{S}(n)$ and $\mathcal{A}(n)$ in the 100 trials.



(a)

		Δ	*Final*
weakening	M	0.0170	0.5455
	SD	0.03535	0.23710
	Min	-0.07	0.05
	Q1	0	0.3901
	Q2	0	0.6262
	Q3	0.0413	0.6608
	Max	0.14	1
strengthening	M	0.2191	0.7476
	SD	0.35311	0.19165
	Min	-0.47	0.37
	Q1	-0.0005	0.6224
	Q2	0.2344	0.7539
	Q3	0.2756	0.8688
	Max	0.89	1
alteration	M	0.2881	0.8166
	SD	0.28858	0.13299
	Min	0	0.55
	Q1	0.0656	0.6950
	Q2	0.2418	0.8681
	Q3	0.2756	0.8822
	Max	0.89	1

(b)

Figure 5.6: *The original accuracy (red), the accuracy change (blue on the left, Δ on the right) and the resulting accuracy after the revision (black on the left, Final on the right) with the three types of revision. The revision is applied even if the new norm has lower accuracy than the original one.*

Fig. 5.6 compares the accuracy of the original norms with the accuracy of the revised norms, and shows the accuracy change when performing a weakening, strengthening or alteration. As reported earlier in Table 5.7, the accuracy of the original norms was, on average, about 53%, with 75% of the norms having an accuracy below 63%, while the maximum accuracy was 93%. We see that both weakening and strengthening operations in some cases affect the accuracy positively and in other cases negatively. Due to the low number of negative traces, weakening operations were not particularly useful and the final accuracy was not significantly different than the original one (an independent-samples t-test to compare the accuracy of the original norms (M=0.5284, SD=0.21915) with the accuracy of the new norms (M=0.5455, SD=0.23710) does not identify a significant difference: t(198)=-0.527, p=0.599). In the case of strengthening, instead, the average improvement of accuracy was around 20% and this led to new norms with a significantly higher accuracy (independent-samples t-test; accuracy original norms (M=0.5284, SD=0.21915); accuracy new norms (M=0.7476, SD=0.2191); t(198)=-7.526, p=0.000), with a large effect size $d_{Cohen} = 1.065$. Finally, with alteration, the average accuracy improved even more, with a large improvement of around 29% ($d_{Cohen} = 1.59$). Note that half of the new norms had an accuracy higher than 86%, and 25% of the norms had an

accuracy higher than 88%. These results show that the norms revised with *DNR* are significantly better aligned with the MAS objectives on the given datasets than the original ones. Note, finally, that, even though the accuracy improved significantly, it was not perfect in all cases. We will discuss further this and other aspects in Sec. 5.6.1.

Q4. How does *DNR* generalize to previously unseen traces?

So far, we did not discuss the case when the dataset Γ is not a complete sample of all possible traces. In case Γ is limited in representing all possible behaviors that could be exhibited, *overfitting* issues may raise. Overfitting occurs when the model that is used to represent some data (in our case the norm being synthesized) fits too closely the data used to construct such model and fails in generalizing to new data. In machine learning, in order to detect overfitting, different datasets are used for training (in our case revising the norms) and testing the model (evaluating the alignment of the revised norms with the MAS objectives). If no dataset other than Γ is available, different techniques can be employed, such as splitting the data in two parts for training and testing (called respectively training and test set).

To answer **Q4** regarding generality, we perform two experiments. In the first experiment (*75-25 splitting*), we use a standard splitting technique: we take the datasets obtained for **Q1** and we split them in a training and a test set, composed by 75% and 25% of the traces in the dataset, respectively. In the second experiment (*Independent test set*), we use the full datasets as training sets, and a new set of traces (obtained by running the highway simulation with no norm enforced) as test set. In both experiments, we apply then our approach for norm revision on the training set, as done per **Q3**. This time, however, we compare the accuracy of the original norm and the revised one on the test set. We focus only on alteration. Table 5.9 reports the results for both experiments.

75-25% splitting. Note, first, that on the training set the revised norms have significantly higher accuracy than the original ones, in line with the results obtained for **Q3** (compare the values here with the values for alteration in Fig. 5.6). Analogous results are obtained also on the test set, composed by previously unseen traces: while the original norms had an average accuracy of about 52.7%, the revised norms had an accuracy of about 81.4%. Comparing the accuracy of the revised norms on the training and test set, we notice that the accuracy is obviously higher on the training set, but the change is minor ($d_{Cohen} = -0.044$) showing that the revised norm generalized well also to previously unseen traces. Since we split the original datasets, however, the traces composing the test sets, while they were unseen during the revision process, they are not completely independent from the traces composing the training sets, as they were both obtained by monitoring the behavior of the agents under the enforcement of the same original norms.

Independent test set. Note again the expected improvement of the accuracy in the training set with the revised norms, as per previous case. If we compare, instead, the accuracy of the revised norms on the training and test set, we observe that this time the accuracy is about 17% lower on the test set ($d_{Cohen} = -0.845$). This

Table 5.9: *Statistics about the accuracy of the original and revised (altered) norms on the training and test sets. In 75-25 splitting, the training and test sets are composed, respectively, by 75% and 25% of the traces in the original datasets. In Independent test set, the training set is the full set of traces in the original datasets, and the test set is an independent set of traces obtained running the simulaition with no norm enforced.*

	75-25 splitting				*Independent test set*			
	Training set		Test set		Training set		Test set	
Norms	Original	Revised	Original	Revised	Original	Revised	Original	Revised
M	0.527	0.8196	0.5271	0.8136	0.526	0.8202	0.519	0.6537
SD	0.21893	0.13246	0.22012	0.13876	0.21842	0.13529	0.25968	0.2436
Min	0.11	0.55	0.09	0.54	0.1	0.55	0.11	0.11
Q1	0.3459	0.6983	0.3533	0.6853	0.3478	0.695	0.3762	0.3762
Q2	0.6192	0.865	0.616	0.8549	0.6238	0.8718	0.6238	0.6238
Q3	0.6368	0.8897	0.6467	0.9	0.6238	0.8862	0.6238	0.8862
Max	0.93	1	0.91	1	0.93	1	0.89	0.89

shows a certain, expected, degree of overfitting to the training set[7]. However, we can note that the revised norms were better aligned than the original norms also on the previously unseen traces composing the test set: while the original norms had an average accuracy of about 52%, the accuracy of the revised norms was about 12.5% higher (corresponding to an intermediate increase $d_{Cohen} = 0.535$).

We conclude that the revised norms were better aligned with the MAS objectives than the original norms, and they also generalized better than the original norms on previously unseen traces.

5.6.1 Discussion

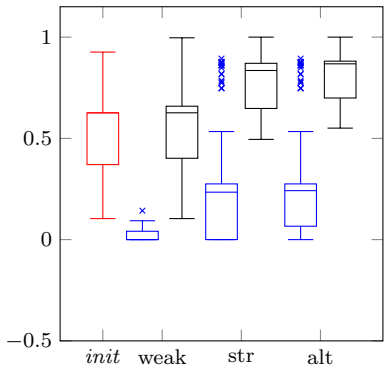
We discuss a number of aspects and some limitations of our proposal and the results above described.

Perfect alignment

In some of our experiments, the norms that were synthesized were less aligned with the MAS objectives than the original ones (e.g., a too weak new norm n' was selected so that, even though all FN traces were correctly classified as positive by n', many bad traces were misclassified as FP). In the experiments this happened in about 20% of revisions for weakening and in about 25% of cases for strengthening (see in Fig. 5.6b the lower quartiles Q1 of the accuracy change for weakening and strengthening, 0 for weakening and negative for strengthening). *In our experiments*, it never happened, instead, to have a negative change of accuracy with alteration (the min value for accuracy change in Fig. 5.6 for alteration is 0). However, similarly to weakening and strengthening, also alteration could exhibit strong side effects and a negative change in some cases. To overcome this possible negative effect, when choosing the new

[7] We do not discuss them here, but in order to mitigate the effect of overfitting, different techniques, such as cross-validation techniques [181], can be employed during the training of a model (in our case, during the revision process).

norm, one could apply additional (or different) criteria to the evaluation provided by a metric like accuracy. A trivial example is discarding new norms that reduce the alignment with the MAS objectives compared to the original norm, and leave unchanged the original norm if none of the new norm is better aligned. Applying this simple additional step to our approach, prevents to revise the original norm into new norms that are less aligned with the MAS objectives[8]. Fig. 5.7 illustrates the effect of adding this simple additional step. Compare the the blue boxes in the figure, with the blue boxes in Fig. 5.6a. Note how now all changes are either positive or 0.



(a)

		Δ	*Final*
weakening	M	0.0201	0.5485
	SD	0.03165	0.23227
	Min	0	0.10
	Q1	0	0.3911
	Q2	0	0.6262
	Q3	0.0413	0.6608
	Max	0.14	1
strengthening	M	0.2633	0.7918
	SD	0.30455	0.14293
	Min	0	0.50
	Q1	0	0.6445
	Q2	0.2344	0.8352
	Q3	0.2756	0.8743
	Max	0.89	1
alteration	M	0.2881	0.8166
	SD	0.28858	0.13299
	Min	0	0.55
	Q1	0.0656	0.6950
	Q2	0.2418	0.8681
	Q3	0.2756	0.8822
	Max	0.89	1

(b)

Figure 5.7: *The original accuracy (red), the accuracy change (blue) and the resulting accuracy after the revision (black) with the three types of revision. The original norm is left unchanged if no norm improves the accuracy.*

However, even though the accuracy increased (compare the black boxes in Fig. 5.7 with those in Fig. 5.6), we still did not obtain a perfect accuracy in all cases. This is due to two reasons: (i) the possibility to perfectly approximate the MAS objectives by means of conditional norms, (ii) the types of norms that can be synthesizes by means of our revision operations. We briefly discuss these two reasons.

As discussed in Sec. 5.4, a (set of) conditional norm(s), in some cases, may not be expressive enough to perfectly represent the MAS objectives. In our simple highway scenario, for example, we are trying to align as much as possible one norm concerning the speed limit of the vehicles in the highway section, with MAS objectives concerning travel time and CO_2 emissions. Regulating only the speed of the cars (and doing

[8]We emphasize, however, that doing so may be harmful in case the dataset of traces used to evaluate the norms is not a good sample of the set of all possible behaviors, as it may increase the chances of overfitting, as discussed in **Q4**.

it only with one norm), obviously, may not always be sufficient to achieve any possible objective. For example, the CO_2 emitted by a vehicle does not only depend on its speed but also on its acceleration. Norms regulating only the speed of the vehicles, may not be sufficient, therefore, to fully characterize (and achieve) the MAS objectives. In this chapter, we discussed how to revise a given set of norms w.r.t. a certain dataset of traces. We did not discuss the need of more or different types of norms when the considered norms are not enough expressive, as this goes beyond the scope of this work. We briefly show here, however, that if the MAS objectives can be expressed with a single conditional norm in the same language of the norm being revised, our heuristic tends to perfect approximation when enough data is provided[9].

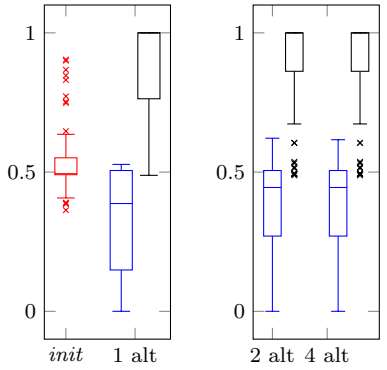
Consider the 100 different datasets of traces from the earlier experiments. We re-label the traces in the datasets. Instead of considering MAS objectives concerning the CO_2 emissions and the travel time of the vehicles, we consider a new MAS objective that concerns the speed of cars and that can be expressed by a conditional prohibition. In particular, we re-label each trace as positive if the vehicle maintains a speed below 70 km/h between the 3th and 9th km of the highway section, and as negative otherwise. We revise then the enforced norm and analyze the change in accuracy with the new norm like we did before. Fig. 5.8a shows the accuracy change when performing one alteration of the norm. The accuracy of the original norms ranged around 0.5. Notice how after one alteration the accuracy is already 1 in more than 75% of cases, while in Fig. 5.6 the accuracy is 1 only in *one* case. Even though we significantly improved the alignment compared to the previous example, the norms are still not perfectly aligned in all the cases: in about 25% of the cases the accuracy is still lower than 0.76.

This brings us to the second reason why a perfect alignment is not always reached: the types of norms that can be synthesized w.r.t. a given dataset. *DNR* determines approximate norm revisions as a more practical solution than exhaustively searching the space of all possible norms, which as shown is intractable. In order to do so, as seen in Sec. 5.5, our heuristic revision operations rely on:

- The propositions that hold in the states of traces that are in the dataset. If the dataset is not informative enough, for example if it contains very few negative traces (e.g., if the dataset was obtained by enforcing a very strict norm, forcing agents to go very slow and causing a jam), the set of revised norms that can be obtained by applying our heuristic, which relies on the available data (i.e., on the content of the states of the traces in the dataset), is limited, and may not contain a new norm that is perfectly aligned with the MAS objectives.

- The set Φ. As mentioned in Sec. 5.5.1, in our experiments we only selected a fixed number of propositions to be considered during the *synthesis step*. In

[9]Note that if MAS objectives and norms can be both expressed in the same language, the setting is closer to the setting from Corapi *et al.* [94] where, as described in Sec. 5.2. Yet, the setting is different. Even though we are now assuming that MAS objectives can be perfectly expressed by means of conditional norms, we still do not have any knowledge about *why* a certain trace is positive or negative (i.e., which parts of the states of a trace make the trace positive or negative), as it is the case in the mentioned work. Instead, our revision mechanism still solely relies on the given boolean labeling of the traces.



(a) (b)

	1 alt		2 and 4 alt	
	Δ	*Final*	Δ	*Final*
M	0.33	0.86	0.37	0.90
SD	0.19	0.19	0.18	0.17
Min	0	0.49	0	0.49
Q1	0.14	0.76	0.27	0.86
Q2	0.39	1	0.44	1
Q3	0.51	1	0.51	1
Max	0.53	1	0.62	1

(c)

Figure 5.8: *The original accuracy (red), the accuracy change (blue) and the resulting accuracy of the revised norm (black) after one alteration (a) and after two and four alterations (b). The change reported in blue refer to the accuracy of the original norm. Table (c) reports the detailed statistics about the plots in (a) and (b).* Δ *refers to the accuracy change, and Final refers to the revised norms.*

particular we selected up to eight[10] higher or lower speeds for the prohibited state, and positions for the condition and deadline. Consider, as an example, an original norm prohibiting vehicles to have a speed higher than 10 km/h. When generating more specific speed limits, we limited the maximum number of more specific speed limits to 8 (for example the speeds 11, 15, 16, 20, ..., 24). If for instance the MAS objectives were not achieved when a vehicle was speeding over 40 km/h, but they were achieved when a vehicle was having a speed of 30 km/h, due to the limitations imposed to the set Φ, we would not generate a new norm that could perfectly distinguish the negative and positive traces. One way to try to mitigate this effect is by revising a norm more than one time. By revising again the new norm, we may be able to correct also some (possibly new, in case of alteration) errors that could not be captured only with one revision, like in the above example. Starting from an original norm n, we can alter it into a new norm n' and then we can alter n' into a further alteration n'', etc.. Fig. 5.8b and the last column of Fig. 5.8c report the effect of doing so in our simple example after two and four consequent alterations. After two alterations, we further improved the accuracy of some of the resulting norms. Note that this monotonic behavior is exhibited because when we are applying the revision operations we are considering again the same dataset of traces Γ. With more than 2 revisions, instead, we did not obtain any further change. Even though more than 75% of traces had an accuracy higher than 86%, with an average of

[10]The value of this parameter was obtained empirically by running a number of preliminary experiments in the highway scenario. Even though we did not notice particularly significant differences, due to the simplicity of the domain, in our preliminary tests, values higher than 8 did not provide any advantage, while with lower values the generated norms, due to their restricted number, provided in some few cases lower improvements.

90%, in about half of the cases the revised norms were still not perfectly aligned with the MAS objectives.

Multiple Norms

As discussed in Sec. 5.5.4, *DNR* also supports the revision of multiple norms. We briefly report some experiments performed on our highway scenario enforcing *two norms* instead of one. Note that to do justice to the complexity of enforcing multiple norms in a MAS, a number of additional aspects (e.g., the importance of the different norms, their mutual constraints, their conflicts, properties, synergies, etc.) shall be taken into account, considering also a complex enough case study. While these aspects extend our work, they are out of the scope of this chapter. We leave for future work, therefore, an in-depth evaluation of the multiple norms case. We limit ourselves here to a preliminary experimentation, which, however, supports, experimentally, the claims made in Sec. 5.5.4, and shows that *DNR* can effectively revise multiple norms and improve their alignment with the MAS objectives.

Similarly to the case of one norm we adopt the following experimental method.

1. We execute a simulation of the highway scenario with two norms. In addition to the norm n_s concerning the speed limit, we now enforce also a norm that regulates the minimum safety distance that vehicles shall maintain in the highway. The minimum safety distance prohibition $n_d = (\phi_C, P(\phi_P), \phi_D)$ is such that ϕ_C, ϕ_P, ϕ_D are boolean combinations of propositions from the languages $L_C = VP \cup VT$, $L_{Pdist} = VD \cup VT$ and $L_D = VP$, respectively, with VP and VT defined as per Sec. 5.3.1 and $VD = \{dist_x \mid 0 \leqslant x \leqslant 15 \;\&\; x \in \mathbb{N}\}$, with literals $dist_x$ denoting that vehicle's distance from the vehicles ahead is lower than x meters[11]. An example of minimum safety distance prohibition is $n_d = (km_2 \wedge car, P(dist_{10}), km_7)$, which prohibits *cars* to maintain a distance smaller than 10 meters from the vehicles ahead from km_2 to km_7 of the highway section. As per n_s, in each simulation, norm n_d is randomly selected from the set of norms defined by its language.

2. The simulation generates a dataset Γ containing 1500 traces labeled w.r.t. the obedience of each norm and the achievement of the MAS objectives of the running example. Note that the i-th state of a trace now, in addition to the information described for Eq. 5.5, contains also information about the minimum distance the agent maintained from any vehicles ahead in the i-th km of the highway.

3. Given Γ, we perform the *synthesis step*. We focus only on alteration here for brevity. We obtain, thus, for each norm, a set of revised norms: $\mathcal{A}(n_s)$ and $\mathcal{A}(n_d)$.

4. We perform the *selection step* to select from $\mathcal{A}(n_s)$ and $\mathcal{A}(n_d)$ the revised norms. We repeat this step two times in two different experiments—the subject of our

[11] Note that, as they do for their speed, vehicles autonomously adjust also their distance from the vehicle ahead based on their internals. We do not directly affect their decisions about the specific safety distance to maintain. Instead we regulate the *minimum* safety distance.

analysis—that use two different strategies to evaluate and select the norms: (*independent*) we select, by using the *acc* metric, each norm from its corresponding set, independently form the other norm; (*combined*) we compare all combinations of norms from the two sets, and we select the two revised norms by using the multi-label accuracy metric *ml-acc* defined in Eq. 5.4.

5. We repeat 100 times the steps 1-4, to obtain a statistically significant comparison. The results that we report refer to such 100 repeated trials.

Table 5.10 reports a comparison of the original and revised norms, for the two experiments of independent and combined revision of the norms. In column *independent*, values concern the average *acc* of the norms, and in column *combined*, values concern the value of *ml-acc*. By comparing the mean values of the original and revised norms,

Table 5.10: *Statistics about the accuracy (acc) and multi-label accuracy (ml-acc) for the original and revised (altered) norms in the cases of independent and combined selections.*

Experiment	*independent*		*combined*	
Metric	*acc*		*ml-acc*	
Norms	Original	Revised	Original	Revised
M	0.5137	0.7543	0.4856	0.6608
SD	0.24513	0.10807	0.27458	0.16833
Min	0.03	0.51	0	0.04
Q1	0.3186	0.6891	0.2390	0.6166
Q2	0.6225	0.7512	0.6201	0.6416
Q3	0.6382	0.8035	0.6379	0.7523
Max	0.90	0.99	0.89	0.99

we observe that in both experiments the alignment with the MAS objectives (i.e., the average accuracy of the two norms in *independent*, and the the multi-label accuracy in *combined*) significantly[12] improved after the revision of the norms. This effect was large ($d_{Cohen} = 1.27$) for the independent revision, and intermediate ($d_{Cohen} = 0.769$) for the combined revision.

In Sec. 5.5.4, we discussed the difference between an independent and a combined revision of the norms in terms of partly and fully correctly classified traces. We argued that independent revisions are more suitable for cases when it is less important for the traces to be classified correctly by all norms, while combined revisions are more suitable when it is important to have fully correct traces. We conclude this section by briefly illustrating this concept in our experiment. Fig. 5.9 reports three confusion matrices: (a) for the original norms of our experiment, (b) for the revised norms obtained in the *independent* experiment, and (c) for the revised norms obtained in

[12]We conducted two independent-samples t-test to compare the change of the alignment of the original and revised norms. For the *independent* revision we have: original norms (M=0.5137, SD=0.24513), revised norms (M=0.7543, SD=0.10807), and we identify a significant difference; t(198)=-8.980, p=0.000. For the *combined* revision we have: original norms (M=0.4856, SD=0.27458), revised norms (M=0.6608, SD=0.16833), and we identify a significant difference; t(198)=-5.439, p=0.000.

the *combined* experiment. Note that the values in the confusion matrices are the mean values for the 100 different norms (e.g., the top left value in Fig. 5.9a, is the mean number of *Positive Fully Correct* traces in the 100 original norms). For the sake of readability, in this last informal discussion, we omit from the figure the standard error of the mean values.

objectives		n_s *ob*		n_s *viol*	
		n_d *ob*	n_d *viol*	n_d *ob*	n_d *viol*
	true	701.11	3.34	46.48	0.69
	false	662.36	12.55	71.72	3.08

(a)

objectives		n_s *ob*		n_s *viol*	
		n_d *ob*	n_d *viol*	n_d *ob*	n_d *viol*
	true	620.75	12.98	95.63	22.26
	false	115.85	32.54	320.43	280.89

(b)

objectives		n_s *ob*		n_s *viol*	
		n_d *ob*	n_d *viol*	n_d *ob*	n_d *viol*
	true	697.79	1.43	28.13	24.27
	false	309.90	5.97	154.28	279.56

(c)

Figure 5.9: *Confusion matrices of the original norms (a), revised norms obtained in the independent experiment (b), and revised norms obtained in the combined experiment (c). Values in the matrices are the mean values for the 100 different norms.*

Fig. 5.9a clearly highlights how the original norms, on average, fully correctly classified almost all the positive traces (top left value), while they fully wrongly classified almost all the negative traces (bottom left value). This shows that the original norms (both of them) were generally too weak and permitted too many negative behaviors. Comparing matrices Fig. 5.9b and Fig. 5.9c, we find a confirmation of the above discussion. Both the independent and combined approaches improved the alignment of the norms with the MAS objectives, reducing the fully wrongly classified negative traces (bottom left value) and improving the fully correctly classified negative traces (bottom right value). With the *independent* revision, we observe however also a lower number of fully correctly classified positive traces (top left) and a higher number of partly correctly classified traces (the four central values). With the *combined* revision, instead, we can observe a higher number of fully correctly classified positive traces (only slightly reduced compared to the original norms), and, while the number of fully wrongly classified negative traces was higher than with the independent approach, the revised norms had lower number of partly correctly classified traces (compare the four central values of the two confusion matrices).

5.7 Threats to Validity

We discuss the main threats to the validity [295] of our work. We mostly focus on the part concerning the heuristic and the experiments that we conducted. All the proofs

have been double-checked by all authors.

Construct validity. The implementation of our heuristic operations and of the highway scenario simulation could be incorrect, which would render the results invalid. We reduced the potential impact of this threat by performing an extensive testing of our implementation and by relying for the simulation engine on the SUMO traffic simulator, a state-of-the-art simulator widely employed and tested independently by different parties due to its open-source nature. Furthermore, to mitigate this threat, in Sec. 5.6, among other things we analyzed the sets of norms synthesized by our operations, explicitly discussing their characteristics w.r.t. the expectations from the heuristic described in Sec. 5.5.

Internal validity. To minimize the threats to internal validity, all our experiments were conducted on a set of datasets obtained by randomly sampling the space of all possible norms. Furthermore, when we evaluated different metrics, we compared the results with a control experimental condition in which we randomly choose the new norm, instead of using a particular metric.

External validity. The reported experimental results are subject to the language used for representing the norms and to the type of dataset obtained from the highway scenario simulation. To mitigate this threat to the external validity, we paid attention to our interpretation and the wording of the implications. We emphasize, however, that the heuristic that we proposed is not derived from the highway scenario, which was instead considered just as a use case to proof our concept. Different use cases should be considered to further confirm our results. Furthermore, in Sec. 5.5.5, we had shown that the number of comparisons that need to be performed to choose a new set of norms, grows exponentially with the number of norms being revised, if the norms are not independent. We did not provide an empirical evaluation for instances of the problem of growing size, and this is a threat to the external validity. Further experiments are clearly required in this sense. In order to mitigate this issue, however, different strategies could be adopted. For example, when norms are inter-dependent, it may be known how the obedience of a norm influence the obedience of another norm. If this information is known or learned [293] it could be used to help choosing between two norms to revise: it is enough to revise one of them to obtain an effect on the other.

5.8 Conclusions

We investigated the problem of norm revision in contexts where agents' internals are unknown and where knowledge about the relationship between the enforced norms, the agents' behavior and the MAS objectives solely relies on the monitored system's execution. We presented results regarding the revision of conditional norms (prohibitions and obligations) with deadlines w.r.t. a set of observed traces. The traces are partitioned into positive and negative ones, depending on whether they help or hurt MAS objectives which, besides their boolean evaluation, are opaque to the revision mechanisms. We demonstrated the NP-completeness of the problem of revising (including alteration, weakening and strengthening) conditional norms. The results motivated us to propose *DNR* (Data-driven Norm Revision): a more practical heuris-

tic approach to obtain approximate revisions of the conditional norms.

DNR consists of two steps: the *synthesis step* and the *selection step*. The synthesis step generates a set of candidate new norms by revising the original norms based on the given dataset of execution traces. The selection step selects the final norms from the synthesized set. In doing so, a norm is interpreted as a binary classifier distinguishing obeying and violating execution traces. Statistical metrics, such as accuracy or F-measure, are used to evaluate and select the best norm among the possible candidates w.r.t. the classification of the traces provided by the MAS objectives. We applied *DNR* to a traffic simulation, and we studied the alignment of the revised norms with the MAS objectives. Results show that the revised norms are significantly better aligned with the MAS objectives than the original norms, exhibiting, in the case of accuracy, an average improvement on the given dataset of traces of about 30% and an average improvement of about 13% on previously unseen traces. We compared the norms selected when using a number of different statistical metrics in the *selection step*, showing for instance that some metrics (e.g., metrics based on the concept of entropy) tend to select more "extreme"–i.e., very weak or very strict–revisions than others, and we provided a comparison in terms of their precision and recall.

In addition to the future directions outlined in Sec. 5.6, we intend to embed *DNR* in the run-time supervision framework presented in earlier chapters that continuously monitors the system's execution and, based on probabilistic strategies, suggests how to revise the norms to continuously guarantee the achievement of the MAS objectives. Incorporating a degree of norm violation, and altering the sanctions, is another way for steering the MAS behavior. Experimentation on multiple cases in different domains is also necessary to identify algorithms that perform well in different MAS types. Finally, in defining the norm revision operations, we treated any state and any proposition as equally important. In some cases, however, certain states or propositions may be more important then other and, in revising norms, one would expect to be able to consider such aspect. This is another future direction of our work.

6 Conclusion

In this thesis, we presented a novel framework for the supervision of autonomous software systems. Our framework aims at the supervision of modern complex systems which operate in highly dynamic, ever-evolving and weakly controllable environments. Examples of these systems include Socio-Technical Systems (STSs), heterogeneous systems where the involved actors are both technical (software) and social (humans and organizations). Smart roads, smart homes, or social media sites are examples of STSs. Actors in a STS are autonomous, can enter or leave the system at any moment, and continuously interact with each other according to their own preferences, and influenced by their dynamic social relationships.

In a smart road, for instance, the behavior of autonomous vehicles depends on their own preferences and goals, which reflect the preferences and goals of the stakeholders and users they operate for. Their internals are not necessarily aligned with, nor known to, the other actors in the system, like other vehicles, pedestrians, smart traffic lights, speed cameras, nor to the designer of the system (e.g, the city council). The opacity of their internals is what makes them weakly controllable. A smart road system, moreover, is part of a broader dynamic and evolving environment, and its operations are continuously affected by natural events, such as storms, snow, etc., but also by social events, such as a football matches, rush hours, or changes in the national regulations.

Despite their dynamism and complexity, software systems are expected to continuously ensure that the overall system's objectives are met. For instance, a smart road infrastructure is expected to guarantee adequate throughput and CO_2 emissions. During the design of a system, however, software engineers are forced to make assumptions, which may become invalid over time, due to the ever-evolving context in which the system operates. Designed speed limitations in a road network, for example, may become inappropriate if new national regulations about the maximum levels of CO_2 emissions are put in place in order to cope with the global warming problem. When design-time assumptions are invalidated, previously defined requirements may become inadequate to guarantee the achievement of the (possibly new) system's objectives, and they need to be revised.

In this thesis, we proposed a framework that provides a system with the capability of automatically revising its requirements at run-time without requiring continuous

human intervention. In particular, our framework is data-driven, in the sense that it continuously monitors the execution of the system, and intervenes by deciding which requirements should be revised when there is evidence from data that the current ones are not effective in ensuring the achievement of the system's objectives.

This dissertation contributes to the disciplines of Artificial Intelligence and Software Engineering. More specifically, the thesis contributes to the research areas of (Normative) Multi-Agent Systems and Requirements Engineering, in the context of autonomic computing.

Our work is characterized by three main pillars, which distinguish it from the state-of-the-art literature.

- *Requirements shall regulate the behavior and interactions of the (social) actors in the system.* Actors of modern software systems are autonomous agents, and their internals are generally opaque to, and weakly controllable by, the system's designer, as they often belong to different stakeholders who have independent control over them. In this dissertation, requirements are represented as norms that, similarly to our society, coordinate and regulate the behavior of individuals without over-constraining their autonomy.

- *Valid and effective requirements and norms can be obtained only at run-time and by using execution data.* Formally proving at design-time the effectiveness of predefined requirements in guaranteeing the system's objectives in all possible operating contexts, is often impossible in practice. Modeling all possible states of complex and dynamic systems leads to the state explosion problem. The openness of modern systems and the autonomy of the involved actors, moreover, make it hard to predict and model at design-time all possible interactions among the actors. In our proposal, the validity and effectiveness of requirements and norms is assessed at run-time and with respect to execution data in different operating contexts.

- *Requirement and norm revision when adaptation of the system's components is not an option.* Adaptation and re-configuration of the system's components to restore their compliance with the requirements is not always an option. In addition to the lack of control over the internals of the agents, in dynamic environments, the objectives of the system are subject to continuous change. Compliance with static predefined requirements may not suffice to ensure the new objectives. We consider the revision of requirements and norms when data provides evidence that the current ones are not effective.

In the following, we revisit the research questions outlined in Chapter 1, and we present our conclusions.

6.1 Answering the Research Questions

RQ 1. *How to validate at run-time the design-time assumptions that are reflected in the requirements?*

The requirements of software systems are often represented by means of requirements models [40, 271]. Requirements models organize the elicited requirements in a hierarchical tree-like structure, where the high-level objectives of the system are refined in terms of more specific requirements and system functions. When specifying requirements and requirements models, engineers make both explicit and implicit assumptions based on the available domain knowledge. Such assumptions, thus the requirements, may become invalid at run-time when the system's operating context changes.

We answered **RQ 1** in Chapter 3, where we studied how execution data can be used to validate, at run-time and in different operating contexts, a variety of design-time assumptions. We considered assumptions concerning the contextualization of requirements, and the relationship and synergy between different requirements and between requirements and the system's objectives. For instance, we considered the assumptions that requirements are satisfied in given contexts, that the satisfaction of requirements depends on the satisfaction of their sub-requirements in the requirements model, and that the system's objectives are met when the requirements are satisfied.

To answer the research question, we proposed to use a probabilistic model (namely, a Bayesian Network [240]). In particular, we formally discussed how to automatically map a requirements model into a Bayesian Network, whose structure reflects the structure of the requirements model, and extends it with nodes concerning monitorable environmental properties characterizing the system's operating context. We discussed how to use the Bayesian Network as a run-time counterpart to the requirements model. We showed how to learn from system execution data statistical correlations between the satisfaction of the requirements and the achievement of the system's objectives in different operating contexts. We discussed how to use the learned knowledge (i.e., the trained Bayesian Network), to provide a quantitative estimation of the degree of validity of design-time assumptions. Finally, we showed that the trained Bayesian Network can be used not only for informing the automated decision making, necessary in our run-time requirements revision framework, but also to offer the possibility of manual inspection and analysis of the assumptions by the practitioners, thanks to its graphical nature which makes it transparent and explainable [58].

RQ 2. *Which are possible data-driven requirements revision policies?*

In a requirements revision framework, the revision should be guided by some policy that indicates how the requirements should be revised, and in which cases. In a data-driven framework, such a policy makes use of execution data to guide the decisions.

We gave answers to **RQ 2** in Chapters 3 and 4, where we proposed and compared a variety of general and domain-independent data-driven requirements revision policies. The proposed policies are driven by data in the sense that they rely on the Bayesian Network mentioned above, which learns from execution data the statistical relationships between the different requirements and between the requirements

and the system's objectives in different operating contexts. We discussed which are possible data-driven policies to suggest whether some of the requirements should be relaxed, made more strict, or altered in another way, in order to re-align them with the system's objectives.

In our framework, we do not assume the requirements to be effective. In particular, requirements are conceived as norms that the agents in the system autonomously decide to obey or to violate. Data could show us that certain requirements are effective when obeyed in some contexts, but ineffective, or even harmful, in others. The studied policies distinguish between requirements that, according to execution data, are more effective in certain operating contexts when obeyed, and requirements that are more effective when violated. The general idea of our proposed policies is that, if data shows that there is a positive synergy between the satisfaction of the requirements and the achievement of the objectives (as typically desired), we expect to improve the achievement of the system's objectives by reducing the violations of the requirements (e.g., by altering the requirements, or making their enforcement more strict). If, conversely, data shows a negative synergy, we expect to improve the achievement of the system's objectives by increasing the violations of the requirements (e.g., by relaxing or temporarily ignoring them).

In Chapter 3, we discussed revision policies in the context of the revision of requirements models. In Chapter 4, we discussed additional data-driven policies that can be used to determine the (positive or negative) change in the number of norm violations estimated to be necessary to guarantee the achievement of the system's objectives. We showed that such policies can be used to effectively guide the revision of sanctions when used as a means to enforce requirements on weakly controllable agents.

RQ 3. *Which are possible operations to revise requirements at run-time?*

Our framework clearly separates between the *policies* that suggest what type of revision is necessary for the requirements, and the *revision operations*. This separation gives us the possibility to evaluate the two steps in separate phases, but also to support the decomposition of the framework in modules which could be used separately. For example the requirements revision policies could be independently used as a guideline for the software engineers, in cases when automated revision is not possible, and human intervention is required.

Since the main objective of this dissertation, however, is the automated revision of the requirements, we also studied which are possible automatic operations to revise the requirements (norms) based on the type of revision indicated by the revision policy. We gave answers to **RQ 3** in Chapters 3-5, where we proposed novel algorithms to automatically revise both requirements models and specific norms based on the direction provided by the revision policy.

In Chapter 3, we discussed possible operations to identify more relaxed, more strict, or simply different requirements variants of the given requirements model.

In Chapter 4, we first studied what are the characteristics of the preferences of different types of rational agents. We discussed how to combine the statistical information acquired at run-time with an estimation of the preferences of the agents operating in the system to identify appropriate sanctions in line with the suggestion

provided by the revision policies.

In Chapter 5, finally, we considered a type of norms, conditional norms with deadlines, used in NMASs to characterize and regulate patterns of behaviors [10]. Conditional norms include conditional prohibitions and obligations, and they are characterized by three components: a condition of applicability, the main content of the norm, and a deadline. We provided a characterization of a norm as a binary classifier [146]. A perfect classifier (norm) is one that classifies as violation all and only the agent's behaviors–execution traces–that are bad for the achievement of the system's objectives. We studied the complexity of the problem of revising conditional norms in order to improve their alignment with the system's objectives with respect to a dataset of execution traces. We proposed then tractable algorithms to automatically revise all the components of a norm. In the smart road scenario, our revision operations could, for example, refine the given norms so that the new norms apply only to specific types of vehicles, or they could extend the validity of a speed limit for x additional kilometers, so to obtain new norms that are better aligned with the system's objectives and are expected to improve their achievement.

RQ 4. *How well does the proposed run-time requirements revision framework perform?*

We evaluated our framework, the proposed data-driven requirements revision policies, and the revision operations, to verify its effectiveness in determining at run-time new requirements that ensure the achievement of the system's objectives. We conducted our experimentation by means of simulations in the traffic domain. We implemented a prototype of our framework as a supervisor envisioned to be embedded in a smart road infrastructure, aimed at determining adequate traffic rules (requirements) to be enforced in order to guarantee system-level objectives concerning the throughput, the safety of the road, and the levels of CO_2 emissions. We built on the advanced state-of-the-art of traffic simulators such as SUMO [184]. Doing so allowed us to validate the hypothesis of our experiments with simulations that are realistic for smart roads scenarios involving autonomous vehicles.

To answer **RQ 4**, in all main chapters we considered multiple criteria for evaluating our proposals, including the degree of achievement of the system's objectives with the identified requirements, the speed in identifying requirements that ensure the achievement of the objectives, the stability of the system when subject to the revisions, and the resilience to noise in the data coming from sensors. We compared our algorithms with each other and with baseline algorithms not informed by data.

While a detailed evaluation can be found in each of the main chapters of this thesis, the results generally show that our data-driven framework is able to identify effective revisions of the requirements which improve the alignment of the requirements with the objectives and ensure their achievement. The use of Bayesian Networks and of general high-level revision policies, moreover, guarantees good resilience to noise in the data. The revision of the sanctions based on some knowledge about the preferences of the agents, allows to quickly identify effective sanctions that motivate an adequate portion of the population of agents to comply with the norms.

6.2 Future Directions

This thesis opens the doors to several research lines. Many of them, more specifically related to possible refinements and improvements of our proposal, have been already identified and discussed in details in Chapters 3-5. We outline here three main future directions which relate to the thesis as a whole.

Continuous-control of autonomous systems

In this thesis, we employed Bayesian Networks as a probabilistic model to learn from data and to guide the revision of norms and requirements. Our use of *classical* Bayesian Networks was based on the core assumption of consistent behavior of the system over time, so that by monitoring the system for a sufficient amount of time, regularities would emerge. An interesting future direction is to relax such an assumption, and consider cases where the system does not exhibit regularities over time, and its behavior is continuously subject to changes and fluctuations. Relaxing such an assumption generates new challenges related to the continuous-control of the system.

A starting point to extend our framework in this direction, could be the use of Dynamic Bayesian Networks (DBNs) [240]. A DBN is a direct extension of a Bayesian Network and allows to model how variables in the network influence each other *over time*. DBNs generalize Hidden Markov Models and Kalman filters, probabilistic models often used to continuously predict, optimize and control motions and trajectories [152]. In a context of continuous-control, they could be used to continuously predict over time short-term changes in the operating environment, and in the behavior of the agents in the system, and to guide a continuous adjustment of the requirements.

The policies and revision operations proposed in this dissertation, then, should be re-evaluated and re-assessed under the different assumptions. Eventually, they should be extended to take into account that the requirements revision should have a more immediate effect on the behavior of the agents, or to consider the trends identified in the fluctuations. Similarly, in case of irregular behaviors, the space of possible alternative revised requirements at each time step may be different. The hill-climbing approach and the statistical analysis of the execution traces that we adopted in this thesis to guide the exploration of such space, may become inadequate. More dynamic techniques, such as Entropy Regularization [158], or Dynamic Decision Networks [240], could be considered in order to support more dynamic policies and decisions over the revisions to perform.

Inclusion in a broader normative framework and social simulation

One interesting future direction is the inclusion of the proposed supervision framework as part of a broader normative context, where the agents are not only norm-aware but also organization-aware [279], and where also social norms [45] have a place in their decision-making, and in the decision-making of the institution enforcing the regulative norms.

While our framework supports the specification of different norms for different types of agents, we did not focus on the organizational structure of the system and on the roles that agents can have within the organization. Furthermore, we did not explicitly account for the presence of social norms within a population of agents, but we assumed that social norms would be exhibited in the regularities of the behavior of the agents.

In the area of MASs, the Multi-Agent-Based Simulation (MABS) community is an active research community focused on the simulation of social and intelligent behaviors in complex social systems. Simulation is a powerful methodology for researchers and practitioners for predicting and explaining behaviors, for exploring and testing hypothesis, and for designing effective systems. The growing computational power of modern computer systems, and the incredible availability of data that characterizes our times, are finally making (social) simulation possible in large scale, and valuable for our society. This is also shown by the growing attention given to computational social science and computational social choice in analyzing and studying social phenomena, including for instance epidemic spreading phenomena.

Despite simulation has been extensively used to study the dynamics of norms, the focus has been primarily given to norm emergence, spreading, or recognition, providing solutions for the evolutionary analysis of a society of agents. The use of simulation to investigate norm revision from an institutional point of view, however, is still extremely limited [78]. In this thesis we made a step in this direction. An explicit study of the relationship between the enforcement and revision of regulative norms, as per our framework, and the emergence and evolution of social norms in a structured society of agents is an exciting future direction for this work.

How do agents with different roles within a society interpret and internalize the norms enforced by the institution? Does this have an impact on the emergence of new social norms, or on the evolution of existing ones? How can an institution aimed at supervising software systems take into account the (possibly evolving) social norms and the roles of agents when enforcing and revising regulative norms?

These are only few of the questions that need to be addressed in this direction. Answering them requires to consider more complex models of agents and norms than the ones considered in this dissertation, as well as refined norm revision policies and algorithms, so to take into account the structure and values of the society of agents. This future work, however, has potential interesting implications in areas such as policy making [126], economics, epidemics, social coordination, or emerging fields like the one of digitally assisted collective intelligence.

Application to real-world settings

In this dissertation, we focused on the traffic domain and we envisioned our supervision framework as part of a smart road infrastructure aimed at guaranteeing desired system-level properties. Besides an experimentation on multiple cases in different domains to assess the generality of our proposal and to identify which algorithms perform better in different domains, an obvious future direction is the application of our proposal in a real-world case study.

Application of our framework to real-world settings poses a number of additional

challenges.

Research challenges. In this thesis we focused on general and domain-independent solutions to drive the revision of the requirements. Our requirements revision policies are based on analysis of the statistical correlation between the satisfaction of the requirements and the achievement of high-level system's objectives, and they abstract away from the details and particular features of different requirements. The ISO/IEC 25010 [165], for example, describes different types of qualities (Non-Functional Requirements) of software, including performance, reliability, security, maintainability, usability requirements, each of them characterized by different peculiarities and purposes. In a real-world setting, our framework should explicitly consider, through all the steps of the revision process, the particular characteristics of different requirements, their relative priorities, and their relationships.

Technical challenges. In a realistic application of our framework, domain-specific knowledge should be integrated and taken into account more explicitly. In Chapter 5, for example, we proposed general operations to revise requirements purely based on execution traces describing the behavior of the agents. We applied our techniques to the revision of traffic rules regulating, for instance, the maximum speed of the vehicles. In a real-world setting, one should not rely exclusively on the execution traces, but should integrate in the revision process also domain-knowledge which can be leveraged during the revision process. For example, information about the maximum road capacity, or the available models of vehicle's dynamics, should take place within our framework. In a realistic application, architecture principles should be explicitly taken into account to guide system's evolution [157]. The impact of a revision of a requirement on the rest of the system, for example, should be assessed before the revision is actually put in place, in order to avoid safety issues and guarantee reliability. The explicit account of architecture principles in the revision strategies proposed in this dissertation is an interesting and motivating future direction to make our proposal closer to real-world application.

Ethical challenges. Finally, this future direction poses also a number of ethical challenges, which should be tackled. For example, are the revised requirements and sanctions fair to all parties involved in the system? How to assess their fairness? Do they guarantee the safety of the involved actors? In this thesis, we have shown that the framework led to effective requirements in few revisions in a "safe" simulation environment, where possibly poor choices of requirements wouldn't be disastrous. Who is accountable in case the autonomous revision framework makes poor decisions and causes system's failure?

A Properties of Rational Agents' Preferences

We report here a formal definition of the properties of the rational agents' preferences described in Sec. 4.3.3 and Sec. 4.4.2.

Proposition 1. *A basic preference* $Pref(a) = (A, \succeq)$ *for an agent* $a \in Ag$ *is*

- *transitive:* $\forall x, y, z \in A$ *if* $x \succeq y$ *and* $y \succeq z$ *then* $x \succeq z$*; and*
- *complete:* $\forall x, y \in A$ *either* $x \succeq y$ *or* $y \succeq x$ *or* $x \sim y$*.*

Proof. Consider a list $AL = (L_1, ..., L_n)$, a set $\mathcal{B} \subset \mathbb{N}$, a set $BL \subseteq \mathcal{B}^n$ and an ordered set $\mathcal{N}$ of n norms. Let $Pref(a) = (A, \succeq)$ be a basic preference and x, y, z be alternatives in A.
(*Transitivity*) Assume that $x \succeq y$ and $y \succeq z$. We prove the transitivity for two cases: either the preference adheres to Def. 10a (case 1) or the preference adheres to Def. 10b (case 2). For both cases, we show that $x \succeq z$.

Case 1: we have that $req\_bud(x) \leqslant req\_bud(y)$ and $req\_bud(y) \leqslant req\_bud(z)$. By transitivity of $\leqslant$, we have $req\_bud(x) \leqslant req\_bud(z)$. Moreover, we have $\forall k, l \in A, \forall B, B' \in BL : k[B] \succ l[B] \Rightarrow k[B'] \succ l[B']$ for both $x \succeq y$ and $y \succeq z$, such that we have it also for $x \succeq z$. Therefore, we conclude that $x \succeq z$.

Case 2: we have either (i) $prop(x) = prop(y) = prop(z)$ or (ii) $prop(x) \neq prop(y)$ and either $prop(y) = prop(z)$ or $prop(y) \neq prop(z)$. In case (i), we have that $req\_bud(x) \leqslant req\_bud(y)$ and $req\_bud(y) \leqslant req\_bud(z)$, and, by transitivity of $\leqslant$, we have that the first condition of Def. 10b is also satisfied by x and z. In case (ii), instead, we have that $\forall B, B' \in BL : x[B] \succeq y[B']$. If $prop(y) = prop(z)$, we have that $prop(x) \neq prop(z)$ and $\forall B, B' \in BL : x[B] \succeq z[B']$, so the second condition of Def. 10b is satisfied also by x and z. If instead also $prop(y) \neq prop(z)$, then we have that $\forall B, B' \in BL : x[B] \succeq y[B']$ and $\forall B, B' \in BL : y[B] \succeq z[B']$. Take three arbitrary alternatives $x[B], y[B'], z[B'']$. We have that $x[B] \succeq y[B']$ and $y[B'] \succeq z[B'']$. Since the choice of B, B' and B'' was arbitrary, this holds for any possible budget list. Therefore there is no $B, B'' \in BL$ such that alternative $z[B'']$ is strictly preferred to an alternative $x[B]$, i.e., $\forall B, B'' \in BL : x[B] \succeq z[B'']$. Again the second condition of Def. 10b is satisfied by x and z, such that $x \succeq z$.
(*Completeness*) By definition of basic preference, every pair of alternatives $x, y \in A$ has to satisfy either Def. 10a or Def. 10b. Notice that, given x and y, if it is not the

case that $x \succeq y$, then we have that $y \succ x$, therefore for every pair of alternatives $x, y \in A$ either $x \succeq y$ or $y \succ x$.

□

Proposition 2. *A preference $Pref(a) = (A, \succeq)$ for an agent $a \in Ag$ is*

- *transitive: $\forall x, y, z \in A$ if $x \succeq y$ and $y \succeq z$ then $x \succeq z$; and*
- *complete: $\forall x, y \in A$ either $x \succeq y$ or $y \succeq x$ or $x \sim y$.*

Proof. Consider a list $AL = (L_1, ..., L_n)$, a set $\mathcal{B} \subset \mathbb{N}$, a set $BL \subseteq \mathcal{B}^n$ and an ordered set $\mathcal{N}$ of n norms. Let $A = \{(\langle p_1, b_1\rangle, ..., \langle p_n, b_n\rangle) \mid p_i \in L_i \ \& \ (b_1, \ldots, b_n) \in BL\}$ be the set of alternatives over which agents have preferences. Let $A_1, ..., A_k$ be k disjoint subsets of A as per Def. 12, and x, y, z be alternatives in A.
(*Transitivity*) Assume that $x \succeq y$ and $y \succeq z$. If both x, y and z belong to the same A_i for $1 \leqslant i \leqslant k$ then, by Prop.1, $x \succeq z$. Otherwise, if $x \in A_i$, $y \in A_j$ and $z \in A_l$ with $i < j < l$, then, by Def. 12, given $i < l$, $\forall v \in A_i \forall w \in A_l : v \succeq w$, therefore $x \succeq z$.
(*Completeness*) By Prop.1, for every pair of alternatives $x, y \in A_i$ for $1 \leqslant i \leqslant k$, either $x \succeq_i y$ or $y \succeq_i x$. Furthermore, by definition of preference, for all $x \in A_j$ and $y \in A_i$, we have $x \succeq y$, for $1 \leqslant j < i \leqslant k$. We have therefore that for every pair of alternatives $x, y \in A$ either $x \succeq y$ or $y \succeq x$. □

Proposition 3. *Given an ordered set of norms $\mathcal{N} = \langle N_1, ..., N_n\rangle$, and a set of t agent types $\mathcal{T}$, each type corresponding to a consistent preference (as per Def. 13), increasing the sanction of a norm N_j in $\mathcal{N}$ without changing the sanctions of other norms, does not increase the upper bound of the probability $P(N_{viol})$, i.e., $UB(N_{viol}, \mathcal{N})$, for all N in $\mathcal{N}$.*

Proof. In this work, the agent's preferences are not affected by the preferences of other agents. Since the upper bound $UB(N_{viol}, \mathcal{N})$ of the probability of violating a norm N in the context of a norm set $\mathcal{N}$ is determined by the number of agents with reason to violate N, as per Sec.4.3.2, if Prop.3 holds for one agent type, then Prop.3 must hold also for all agent types. In the following we consider, therefore, one agent type T. Furthermore we assume $\mathcal{N}$ composed by at least two different norms (if only one norm is enforced, Prop.3 is trivially satisfied).

We prove Prop. 3 by contradiction.

Let M be the set of most preferred alternatives to act upon for agent type T in the context of $\mathcal{N} = \langle N_1, \ldots, N_j, \ldots, N_n\rangle$ (as per Def. 14). Suppose we increase the sanction of norm $N_j = (p_j, s_j)$, obtaining $N'_j = (p_j, s'_j > s_j)$. Let now M' be the set of most preferred alternatives to act upon for agent type T in the context of $\mathcal{N}' = \langle N_1, \ldots, N'_j, \ldots, N_n\rangle$.

Suppose, by contradiction, that $UB(N_{viol}, \mathcal{N}') > UB(N_{viol}, \mathcal{N})$ for $N = (p, s) \neq N_j$, with $p \in L_i$ and L_i in AL. This means that, in the context of $\mathcal{N}$, T has no reason to violate N, while in the context of $\mathcal{N}'$, T has reason to violate N (i.e., there exists no alternative $c \in M$ with $viol(c, N)$, while there exists an alternative $c' \in M'$ such that $viol(c', N)$).

In order to modify the set of most preferred alternatives M when increasing the sanction from s_j to s'_j, it must be the case that there exists at least one alternative

$c \in M$ s.t. $viol(c, N_j)$ and $s_j \leqslant b_j < s'_j$ (with b_j budget of the j-th pair in c). If it is not the case increasing s_j to s'_j does not affect T's most preferred alternatives and thus the proposition holds.

Consider the alternative from M with highest budget b_j in the j-th pair. Consider also an $i \neq j$. Let $\overline{a}wx[B] = (\langle p_1, b_1\rangle, ..., \langle w, b_w\rangle, ..., \langle x, b_x\rangle, ..., \langle p_n, b_n\rangle)$ be such alternative, with $w \in L_j, x \in L_i$ and $s_j \leqslant b_w < s'_j$. Let the alternative $\overline{b}zy[B'] = (\langle p'_1, b'_1\rangle, ..., \langle z, b_z\rangle, ..., \langle y, b_y\rangle, ..., \langle p'_n, b'_n\rangle)$ be an alternative $c' \in M'$ such that $c' \notin M$ and $viol(c', N)$ with $z \in L_j, y \in L_i$ and $b_z \geqslant s'_j$. Notice that $\overline{a}wx[B]$ is compliant w.r.t. N and $\overline{b}zy[B']$ is not compliant w.r.t. N, hence $y \neq x$. Notice also that $\overline{a}wx[B] \succ \overline{b}zy[B']$. This is because by Def. 14 we have that, since $\overline{b}zy[B'] \notin M$, $\overline{b}zy[B'] \succeq \overline{a}wx[B]$ iff $\overline{b}zy[B']$ violates a norm N_k but the budget is not enough to pay the sanction. Such N_k cannot be N_j, since $b_z \geqslant s'_j$, and if it's another N_k then $\overline{b}zy[B']$ cannot be also in M' because we only increased the sanction of norm N_j. Therefore $\overline{b}zy[B']$ must be strictly less preferred than $\overline{a}wx[B]$. Furthermore, let $\overline{c}$ be a fully compliant alternative.[1]

We first consider the case of $T = (A, \succeq)$ basic preference as per Def. 10, which can adhere to either Def. 10a or Def. 10b, then we uplift the proof to the preference as per Def. 12.

Basic Preference.

(Case Def. 10a) Since $\overline{a}wx[B] \succ \overline{b}zy[B']$, it holds that, due to Def. 10a, $c[B] \succ \overline{b}zy[B']$ for all alternatives c in A. This means that also a fully compliant alternative $\overline{c}[B]$ is such that $\overline{c}[B] \succ \overline{b}zy[B']$. However, since $s'_j \succ b_w$, after revising s_j into s'_j, there is at least one alternative in M' (i.e., $\overline{c}[B]$) that is strictly preferred to $\overline{b}zy[B']$, because already present in M, and that is compliant to N. Therefore, $\overline{b}zy[B']$ cannot be among the most preferred alternatives to act upon, i.e., $\overline{b}zy[B'] \notin M'$ (contradiction).

(Case Def. 10b) Since $\overline{a}wx[B] \in M \succ \overline{b}zy[B'] \in M'$, by Def. 10b it holds that $\overline{a}wx[B'] \succeq \overline{a}wx[B] \succ \overline{b}zy[B'] \succeq \overline{b}zy[B]$ for all $B' \in BL$, i.e., $\overline{a}wx \succ \overline{b}zy$ regardless of the required budget, including the maximum possible budget in $\mathcal{B}$, $max(\mathcal{B})$. Therefore, since $\overline{a}wx$ is the alternative in M with highest budget in the j-th pair, we have that $s'_j > max(\mathcal{B})$, and no alternative that violates N'_j can be chosen in the context of $\mathcal{N}'$. $\overline{b}zy[B']$ is then compliant w.r.t. N'_j, hence $z \neq w$. By Def. 10, T contains at least another alternative $\overline{a}wy[B']$. If $\overline{a}wy[B'] \succeq \overline{a}wx[B']$, then, according to Def. 10b, $\overline{a}wy[B'] \succeq \overline{a}wx[B']$ for all $B' \in BL$. But if this is the case, we have that $s_j > max(\mathcal{B})$ (otherwise at least one alternative $\overline{a}wy$ with $b_w = max(\mathcal{B})$ is in M and, in contradiction with out hypothesis, T has reason to violate N since $viol(y, N)$). But if $s_j > max(\mathcal{B})$ we have $\overline{a}wx[B] \notin M$ (contradiction). If, instead, $\overline{a}wx[B'] \succ \overline{a}wy[B']$ then $\overline{a}wx \succ \overline{a}wy$ regardless of the budget. By consistency (Def 13), then, we also have $\overline{b}zx \succ \overline{b}zy$. We distinguish 2 cases: (a) $\overline{a}wx \succ \overline{b}zx$, this implies $\overline{a}wx \succ \overline{b}zx \succ \overline{b}zy$, which contradicts $\overline{b}zy[B'] \in M'$, since alternatives $\overline{b}zx$ (compliant w.r.t N'_j) are strictly preferred to $\overline{b}zy[B']$; (b) $\overline{b}zx \succ \overline{a}wx$, this implies that, since $\overline{a}wx[B] \in M$ and $\overline{b}zx$ is compliant w.r.t. both L_j and L_i, then for ev-

[1] We call $\overline{a}$ and $\overline{b}$ the list of propositional atoms that are different from w, x, y, z respectively in $\overline{a}wx$ and $\overline{b}zy$. Also, we use notation $c[B]$ to indicate an alternative c with list of budgets B, as described in Sec.4.3.2.

ery other norm violated by $\overline{b}zx$, the sanction associated to such norm is bigger than $max(\mathcal{B})$ (otherwise $\overline{a}wx \notin M$ and at least one alternative $\overline{b}zx \in M$). But if this is the case, also $\overline{b}zy[B'] \notin M'$, since the only sanction that we change is s_j, and again we have a contradiction.

Preference.

In the case of preference $T = (A, \succeq)$, the k basic preferences composing A adhere to either Def. 10a or Def. 10b. The only case non considered above is when $\overline{a}wx[B] \in A_i, \overline{b}zy[B'] \in A_p$ and it does not exists an alternative $c \in A_p$ s.t. $\overline{a}wx[B] \succ c$, and it does not exists an alternative $c' \in A_q$ s.t. $c' \succ \overline{b}zy[B']$, for two basic preferences A_p, A_q composing A, with $1 \leqslant p < q \leqslant k$. Since all alternatives in A_i have required budget lower or equal than $max(\mathcal{B}_i)$, then, due to Def. 10, among A_p there is at least one fully compliant alternative with required budget $\leqslant max(\mathcal{B}_i)$. Therefore, even if sanction for N_j is increased to a value $s > max(\mathcal{B})$, M' does not contain any alternative from A_q that was not already in M, therefore $\overline{b}zy[B'] \notin M'$, and again we have a contradiction. □

B Full Preferences of Four Types of Agents

We report here the full preferences of the four types of agents considered in the experimentation of Chapter 4, described in Sec. 4.6.

- *BraveRich*:

$$
\begin{gathered}
(\langle sp_{15},0\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{15},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{0.5},0\rangle) \succ \\
(\langle sp_{15},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},0\rangle) \succ \\
(\langle sp_{15},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},1\rangle) \succ (\langle sp_{15},2\rangle,\langle dist_{0.5},2\rangle) \succ \\
(\langle sp_{8},0\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{8},1\rangle,\langle dist_{0.5},0\rangle) \succ \\
(\langle sp_{8},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{0.5},0\rangle) \succ \\
(\langle sp_{8},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{0.5},1\rangle) \succ (\langle sp_{8},2\rangle,\langle dist_{0.5},2\rangle) \succ \\
(\langle sp_{3},0\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{3},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{0.5},0\rangle) \succ \\
(\langle sp_{3},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{0.5},0\rangle) \succ \\
(\langle sp_{3},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{0.5},1\rangle) \succ (\langle sp_{3},2\rangle,\langle dist_{0.5},2\rangle) \succ \\
(\langle sp_{15},0\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{15},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{1},0\rangle) \succ \\
(\langle sp_{15},1\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{1},0\rangle) \succ \\
(\langle sp_{15},1\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{1},1\rangle) \succ (\langle sp_{15},2\rangle,\langle dist_{1},2\rangle) \succ \\
(\langle sp_{8},0\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{8},1\rangle,\langle dist_{1},0\rangle) \succ \\
(\langle sp_{8},1\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{1},0\rangle) \succ \\
(\langle sp_{8},1\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{1},1\rangle) \succ (\langle sp_{8},2\rangle,\langle dist_{1},2\rangle) \succ \\
(\langle sp_{3},0\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{3},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{1},0\rangle) \succ \\
(\langle sp_{3},1\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{1},0\rangle) \succ \\
(\langle sp_{3},1\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{1},1\rangle) \succ (\langle sp_{3},2\rangle,\langle dist_{1},2\rangle) \succ \\
(\langle sp_{15},0\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{15},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{2},0\rangle) \succ \\
(\langle sp_{15},1\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{2},0\rangle) \succ \\
(\langle sp_{15},1\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{2},1\rangle) \succ (\langle sp_{15},2\rangle,\langle dist_{2},2\rangle) \succ \\
(\langle sp_{8},0\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{8},1\rangle,\langle dist_{2},0\rangle) \succ \\
(\langle sp_{8},1\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{2},0\rangle) \succ \\
(\langle sp_{8},1\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{2},1\rangle) \succ (\langle sp_{8},2\rangle,\langle dist_{2},2\rangle) \succ \\
(\langle sp_{3},0\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{3},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{2},0\rangle) \succ \\
(\langle sp_{3},1\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{2},0\rangle) \succ \\
(\langle sp_{3},1\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{2},1\rangle) \succ (\langle sp_{3},2\rangle,\langle dist_{2},2\rangle)
\end{gathered}
$$

- *BraveMiddleClass*:

$$(\langle sp_{15},0\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{15},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{0.5},0\rangle) \succ$$
$$(\langle sp_{15},1\rangle,\langle dist_{0.5},1\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{0.5},1\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{8},1\rangle,\langle dist_{0.5},1\rangle) \succ (\langle sp_{3},0\rangle,\langle dist_{0.5},0\rangle) \succ$$
$$(\langle sp_{3},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{0.5},0\rangle) \succ (\langle sp_{3},1\rangle,\langle dist_{0.5},1\rangle) \succ$$
$$(\langle sp_{15},0\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{15},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{1},0\rangle) \succ$$
$$(\langle sp_{15},1\rangle,\langle dist_{1},1\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{1},1\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{8},1\rangle,\langle dist_{1},1\rangle) \succ (\langle sp_{3},0\rangle,\langle dist_{1},0\rangle) \succ$$
$$(\langle sp_{3},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{1},0\rangle) \succ (\langle sp_{3},1\rangle,\langle dist_{1},1\rangle) \succ$$
$$(\langle sp_{15},0\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{15},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{2},0\rangle) \succ$$
$$(\langle sp_{15},1\rangle,\langle dist_{2},1\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{8},0\rangle,\langle dist_{2},1\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{8},1\rangle,\langle dist_{2},1\rangle) \succ (\langle sp_{3},0\rangle,\langle dist_{2},0\rangle) \succ$$
$$(\langle sp_{3},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{2},0\rangle) \succ (\langle sp_{3},1\rangle,\langle dist_{2},1\rangle) \succ$$
 # from here ordered by budget
$$(\langle sp_{15},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{0.5},2\rangle) \succeq$$
$$(\langle sp_{8},2\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{15},0\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{1},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{1},2\rangle) \succeq$$
$$(\langle sp_{8},2\rangle,\langle dist_{1},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{1},0\rangle) \succeq$$
$$(\langle sp_{15},0\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{2},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{2},2\rangle) \succeq$$
$$(\langle sp_{8},2\rangle,\langle dist_{2},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{2},0\rangle) \succ$$
$$(\langle sp_{15},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{8},1\rangle,\langle dist_{0.5},2\rangle) \succeq$$
$$(\langle sp_{8},2\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{0.5},1\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{8},1\rangle,\langle dist_{1},2\rangle) \succeq$$
$$(\langle sp_{8},2\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{1},1\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{8},1\rangle,\langle dist_{2},2\rangle) \succeq$$
$$(\langle sp_{8},2\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{2},1\rangle) \succ$$
$$(\langle sp_{15},2\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{0.5},2\rangle) \succeq$$
$$(\langle sp_{15},2\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{1},2\rangle) \succeq$$
$$(\langle sp_{15},2\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{2},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{2},2\rangle)$$

- *BravePoor*:

$$(\langle sp_{15},0\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{15},0\rangle,\langle dist_{1},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{1},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{1},0\rangle) \succeq$$
$$(\langle sp_{15},0\rangle,\langle dist_{2},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{2},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{2},0\rangle) \succ$$
$$(\langle sp_{15},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{0.5},1\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{15},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{1},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{1},1\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{1},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{1},0\rangle) \succeq$$
$$(\langle sp_{15},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{2},0\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{2},1\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{2},0\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{2},1\rangle) \succeq (\langle sp_{3},1\rangle,\langle dist_{2},0\rangle) \succ$$
$$(\langle sp_{15},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{8},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{8},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{8},2\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{3},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{3},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{3},2\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_{1},1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{1},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{1},0\rangle) \succeq$$

$$(\langle sp_8,1\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_3,1\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_3,0\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_3,1\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_3,0\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_2,0\rangle) \succ$$
$$(\langle sp_{15},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_8,1\rangle,\langle dist_{0.5},2\rangle) \succeq$$
$$(\langle sp_8,2\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_3,1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_{0.5},1\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_8,1\rangle,\langle dist_1,2\rangle) \succeq$$
$$(\langle sp_8,2\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_3,1\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_1,1\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_8,1\rangle,\langle dist_2,2\rangle) \succeq$$
$$(\langle sp_8,2\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_3,1\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_2,1\rangle) \succ$$
$$(\langle sp_{15},2\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_{0.5},2\rangle) \succeq$$
$$(\langle sp_{15},2\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_1,2\rangle) \succeq$$
$$(\langle sp_{15},2\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_2,2\rangle)$$

- *Cautious*:

$$(\langle sp_3,0\rangle,\langle dist_2,0\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_2,0\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_3,0\rangle,\langle dist_1,0\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_1,0\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_3,0\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{0.5},0\rangle) \succ$$
$$(\langle sp_3,0\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_3,1\rangle,\langle dist_2,0\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_2,1\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_2,0\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_3,0\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_3,1\rangle,\langle dist_1,0\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_1,1\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_1,0\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_3,0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_3,1\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_{0.5},1\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_{0.5},0\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{0.5},0\rangle) \succ$$
$$(\langle sp_3,1\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_3,0\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_2,0\rangle) \succeq$$
$$(\langle sp_3,1\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_3,0\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_1,0\rangle) \succeq$$
$$(\langle sp_3,1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_3,0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_8,1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_8,0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_{0.5},0\rangle) \succeq$$
$$(\langle sp_{15},1\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},0\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},0\rangle) \succ$$
$$(\langle sp_3,1\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_8,1\rangle,\langle dist_2,2\rangle) \succeq$$
$$(\langle sp_8,2\rangle,\langle dist_2,1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_2,1\rangle) \succeq$$
$$(\langle sp_3,1\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_8,1\rangle,\langle dist_1,2\rangle) \succeq$$
$$(\langle sp_8,2\rangle,\langle dist_1,1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_1,1\rangle) \succeq$$
$$(\langle sp_3,1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_3,2\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_8,1\rangle,\langle dist_{0.5},2\rangle) \succeq$$
$$(\langle sp_8,2\rangle,\langle dist_{0.5},1\rangle) \succeq (\langle sp_{15},1\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},1\rangle) \succ$$
$$(\langle sp_3,2\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_2,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_2,2\rangle) \succeq$$
$$(\langle sp_3,2\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_1,2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_1,2\rangle) \succeq$$
$$(\langle sp_3,2\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_8,2\rangle,\langle dist_{0.5},2\rangle) \succeq (\langle sp_{15},2\rangle,\langle dist_{0.5},2\rangle)$$

Obligation Synthesis is NP-complete

The obligation synthesis problem can be equivalently restated as follows:

Instance: a finite set of traces (finite sequences of states) Γ partitioned into negative traces Γ_F and positive traces Γ_T

Question: are there three sets of states X_C (detachment condition), X_O (obligation), X_D (deadline) such that:

Neg For every trace $\rho \in \Gamma_F$, there exist i and j with $i \leqslant j$ such that $\rho[i] \in X_C$, $\rho[j] \in X_D$, and there is no k with $i \leqslant k \leqslant j$ such that $\rho[k] \in X_O$ (there is a detachment state $\rho[i]$, deadline state $\rho[j]$, and no obligation state $\rho[k]$ in between them)

Pos For every trace $\rho \in \Gamma_T$, if for some i and j, $i \leqslant j$, $\rho[i] \in X_C$, $\rho[j] \in X_D$, then there exists k such that $i \leqslant k \leqslant j$ and $\rho[k] \in X_O$.

Theorem 6. *The obligation synthesis problem is NP-complete.*

Proof. The obligation synthesis problem is clearly in NP (guess the sets and check in polynomial time that they satisfy the conditions). To prove that it is NP-hard, we reduce the 3-SAT problem to it (satisfiability of a set of clauses with 3 literals).

Suppose a set of clauses $C_1, \ldots, C_n$ over variables $x_1, \ldots, x_m$ is given. We generate an instance of the obligation synthesis problem such that it has a solution iff $C_1, \ldots, C_n$ are satisfiable (each clause contains at least one true literal).

The idea of the reduction is just like for prohibitions, but now instead of inserting a deadline between s and t in positive traces, we insert an obligation.

We use two auxiliary states s and t, intuitively to serve as the detachment condition and the deadline, and make sure that neither of them is also the obligation. We want to make some subset of $\{v_i : i \in [1, ...m]\} \cup \{u_i : i \in [1, ...m]\}$ to be the obligation (X_O), so that exactly one of v_i, u_i for each i is in X_O. Then $u_i \in X_O$ can encode that x_i is true, and $v_i \in X_O$ that x_i is false, and we can make the encoding work by creating a positive trace corresponding to each clause so that at least one of the literals in the clause should be true (corresponding value is in X_O).

We use notation $(s_1, \ldots, s_k)$ for a trace consisting of states $s_1, \ldots, s_k$.

The set of negative traces contains:

- a 2 state trace (s, t) [this forces either
  $s \in X_C \cap \overline{X_D} \cap \overline{X_O}$, $t \in X_D \cap \overline{X_C} \cap \overline{X_O}$, or
  $s \in X_C \cap X_D \cap \overline{X_O}$, or
  $t \in X_C \cap X_D \cap \overline{X_O}$.

  To rule out the latter two possibilities, we require below that s and t on their own are positive traces.]

- for every variable x_i in the input, a 6-state trace (s, v_i, t, s, u_i, t) [this means that either v_i or u_i are not in X_O, because there is one $[s, t]$ interval that does not contain a state from X_O]

The set of positive traces contains:

- a one state trace (s) [so s cannot be in $X_C \cap X_D \cap \overline{X_O}$]

- a one state trace (t) [so t cannot be in $X_C \cap X_D \cap \overline{X_O}$]

- for every variable x_i in the input, a 4-state trace (s, v_i, u_i, t) [this means that either v_i or u_i are in X_O]

- for each clause C in the input over variables x_j, x_k, x_l, a 5-state trace (s, z_j, z_k, z_l, t) where z_i is u_i if x_i occurs in C positively, and v_i if it occurs negatively.

The reduction is linear in the number of variables and clauses.

We claim that there exists an assignment f of $0, 1$ to $x_1, \ldots, x_m$ satisfying $C_1, \ldots, C_n$ if, and only if, there is a solution to the obligation synthesis problem above where $s \in X_C$, $t \in X_D$, and for every i, $u_i \in X_O$ iff $f(x_i) = 1$ and $v_i \in X_O$ iff $f(x_i) = 0$.

Assume an assignment f satisfying $C_1, \ldots, C_n$ exists. Let $X_C = \{s\}$ and $X_D = \{t\}$. For every i, place u_i in X_O iff $f(x_i) = 1$ and $v_i \in X_O$ iff $f(x_i) = 0$. This produces a solution because: (s, t) satisfies **Neg**; for every i, either u_i or v_i are not in X_O, so (s, v_i, t, s, u_i, t) satisfies **Neg**. Positive traces satisfy **Pos**: either s followed by t does not occur, or there are u_i, v_i between them, and one of them is in X_O, or (for the clause encoding) one of the literals in the clause is true, so for positive x_i it means that u_i is in X_O and **Pos** is satisfied, or for negative $\neg x_i$ it means that v_i is in X_O and again **Pos** is satisfied.

Assume there is a solution to the obligation synthesis problem. It is clear (see the comments next to traces) that any solution should satisfy $s \in X_C \cap \overline{X_D} \cap \overline{X_O}$ and $t \in X_D \cap \overline{X_C} \cap \overline{X_O}$. Since (s, v_i, t, s, u_i, t) is a negative trace for every i, it means that it contains an unsatisfied conditional obligation: either the one detached in the first occurence of s, or the one detached in the second occurrence of s. Note that $t \notin X_C$, so the last occurrence of t does not detach any obligation. This means that for every i, either v_i or u_i is not in X_O (or neither is in X_O). (Assume by contradiction that v_i and u_i are in X_O, then v_i satisfies an obligation detached in s; if it is also in X_C and detaches a new obligation, then that obligation is also immediately satisfied. If v_i is also in X_D, it still satisfies any obligation detached earlier or in v_i itself since it is in X_O. Similarly, u_i satisfies any obligation detached after the first occurrence of t.) For the assignment we are constructing, this ensures that a variable and its negation cannot both be assigned 1.

Since (s, v_i, u_i, t) is a positive trace, then in any solution, for every i, either u_i or v_i has to be in X_O. Together with the argument above, this gives us that exactly one of u_i, v_i for every i is in X_O. Hence we can use the membership in X_O to produce a boolean valuation of variables x_i (1 if $u_i \in X_O$, and 0 if $v_i \in X_O$).

Since for every clause $C = \{\sim x_j, \sim x_k, \sim x_l\}$, the trace (s, z_j, z_k, z_l, t) (where z_i is v_i if $\sim x_i = \neg x_i$, and u_i if $\sim x_i = x_i$) is a positive trace, at least one of z_i is in X_O. This means that the valuation based on the membership in X_O, for any solution, will also produce a valuation that satisfies all the clauses (since at least one literal in each clause will evaluate to 1).

□

D Conditional Norms Revision Experimentation Data

In the following are reported the detailed data about the experiments performed in Sec. 5.6 of Chapter 5 for question **Q2**.

Table D.1: *Statistics about the change of the % of TP, FP, TN, FN in the norms selected according to the metrics of Sec. 5.5.3. Average values are obtained w.r.t. the 100 different norms selected from the sets of possible norms.*

metric		weakening		strengthening		alteration	
		M	SD	M	SD	M	SD
random	Δ_{TP}	0.022	0.03105	-0.3144	0.27213	-0.2184	0.26913
	Δ_{FP}	0.0359	0.05741	-0.3577	0.23336	-0.1833	0.22449
	Δ_{TN}	-0.0359	0.05741	0.3577	0.23336	0.1833	0.22449
	Δ_{FN}	-0.022	0.03105	0.3144	0.27213	0.2184	0.26913
acc	Δ_{TP}	0.0247	0.03559	-0.1414	0.14539	-0.0481	0.08431
	Δ_{FP}	0.0077	0.01514	-0.3605	0.28243	-0.3362	0.30323
	Δ_{TN}	-0.0077	0.01514	0.3605	0.28243	0.3362	0.30323
	Δ_{FN}	-0.0247	0.03559	0.1414	0.14539	0.0481	0.08431
F1	Δ_{TP}	0.026	0.03579	-0.1324	0.14009	-0.019	0.07818
	Δ_{FP}	0.0241	0.0479	-0.2714	0.1874	-0.1532	0.19799
	Δ_{TN}	-0.0241	0.0479	0.2714	0.1874	0.1532	0.19799
	Δ_{FN}	-0.026	0.03579	0.1324	0.14009	0.019	0.07818
mcen	Δ_{TP}	0.0279	0.03708	-0.3912	0.30209	-0.3632	0.32715
	Δ_{FP}	0.0426	0.06289	-0.3907	0.26829	-0.1558	0.22396
	Δ_{TN}	-0.0426	0.06289	0.3907	0.26829	0.1558	0.22396
	Δ_{FN}	-0.0279	0.03708	0.3912	0.30209	0.3632	0.32715
kappa	Δ_{TP}	0.0229	0.03408	-0.1894	0.20785	-0.1276	0.17602
	Δ_{FP}	0.0223	0.04825	-0.3356	0.21074	-0.2744	0.20478
	Δ_{TN}	-0.0223	0.04825	0.3356	0.21074	0.2744	0.20478
	Δ_{FN}	-0.0229	0.03408	0.1894	0.20785	0.1276	0.17602

Table D.2: *Statistics about the 100 revised norms selected by different metrics in the case of weakening. The values in the table average the 100 independent revisions. Values for TP, FP, TN and FN are the percentage w.r.t. the total number of traces (i.e. 1500) in the dataset.*

metric		TP	FP	TN	FN	precision	recall	*acc*	*F1*	*mcen**	*kappa*
	M	0.5075	0.4793	0.007	0.0061	0.5139	0.934	0.5146	0.6158	0.7261	0.0122
	SD	0.28146	0	0.01926	0.01305	0.28632	0.20256	0.27692	0.32326	0.14848	0.07022
	Min	0	0	0	0	0	0	0	0	0.38	-0.09
random	Q1	0.3679	0.3473	0	0	0.376	0.9748	0.3723	0.5405	0.6662	-0.0027
	Q2	0.6262	0.3738	0	0.0007	0.6262	0.999	0.6262	0.7702	0.6854	0
	Q3	0.6475	1	0.004	0.0038	0.6491	1	0.6475	0.786	0.8013	0
	Max	0.99	1	0.13	0.08	1	1	0.99	1	1	0.51
	M	0.5101	0.451	0.0353	0.0035	0.5163	0.9185	0.5455	0.6189	0.7198	0.0265
	SD	0.28462	0.23752	0.05991	0.00773	0.28757	0.23666	0.2371	0.32567	0.1092	0.08852
	Min	0	0	0	0	0	0	0.05	0	0.49	-0.04
acc	Q1	0	0.3234	0	0	0.3775	0.983	0.3901	0.5374	0.6841	0
	Q2	1	0.3738	0	0	0.6262	1	0.6262	0.7702	0.6854	0
	Q3	0.6424	0.5903	0.0475	0.0025	0.6598	1	0.6608	0.7903	0.7802	0.0009
	Max	1	0.95	0.25	0.04	1	1	1	1	0.97	0.74
	M	0.5115	0.4675	0.0189	0.0022	0.5165	0.9836	0.5304	0.6198	0.7393	0.0258
	SD	0.28367	0.26431	0.03818	0.0056	0.28683	0.07379	0.26297	0.32454	0.12533	0.08122
	Min	0	0	0	0	0	0.33	0	0	0.5	-0.01
F1	Q1	0.3806	0.3265	0	0	0.3806	0.9983	0.3806	0.5512	0.6845	0
	Q2	0.6262	0.3738	0	0	0.6262	1	0.6262	0.7702	0.6854	0
	Q3	0.6466	0.6194	0.01	0.0007	0.6576	1	0.6608	0.7903	0.837	0.0009
	Max	0.99	1	0.15	0.03	1	1	1	1	1	0.67
	M	0.5134	0	0.0004	0.0003	0.5138	0.9883	0.5137	0.6188	0.7638	0.0089
	SD	0.28469	0	0.00128	0.00088	0.28508	0.07168	0.28493	0.32419	0.12759	0.06801
	Min	0	0	0	0	0	0.33	0	0	0.64	-0.01
*mcen**	Q1	0.3806	0.3473	0	0	0.3806	1	0.3806	0.5512	0.6854	0
	Q2	0.6262	0	0	0	0.6262	1	0.6262	0.7702	0.6854	0
	Q3	0.6493	1	0	0	0.6526	1	0.6525	0.7897	0.8387	0
	Max	0.99	1	0.01	0	1	1	1	1	1	0.67
	M	0.5084	0.4657	0.0207	0.0053	0.5175	0.9797	0.5291	0.6188	0.7257	0.0327
	SD	0.28051	0.26632	0.03854	0.01277	0.28792	0.07442	0.26141	0.32364	0.13827	0.0946
	Min	0	0	0	0	0	0.33	0	0	0.4	-0.01
kappa	Q1	0.3806	0	0	0	0.3806	0.9956	0.3806	0.5512	0.6831	0
	Q2	0.6262	0.3738	0	0	0.6262	1	0.6262	0.7702	0.6854	0
	Q3	0.6424	0.6194	0.0278	0.0012	0.6598	1	0.6608	0.7903	0.8345	0.0009
	Max	0.99	1	0.15	0.07	1	1	1	1	1	0.7

Table D.3: *Statistics about the 100 revised norms selected by different metrics in the case of strengthening. The values in the table average the 100 independent revisions. Values for TP, FP, TN and FN are the percentage w.r.t. the total number of traces (i.e. 1500) in the dataset.*

metric		TP	FP	TN	FN	precision	recall	*acc*	*F1*	*mcen**	*kappa*
random	M	0.171	0.0856	0.4007	0.3426	0.8188	0.3028	0.5718	0.2774	0.7001	0.1047
	SD	0.24836	0.16569	0.27568	0.28884	0.31974	0.39134	0.25751	0.37662	0.19619	0.28704
	Min	0	0	0	0	0	0	0.02	0	0.02	-0.75
	Q1	0	0	0.2693	0.0052	0.6499	0	0.3736	0	0.6474	0
	Q2	0	0	0.3738	0.3221	1	0.0011	0.5341	0	0.708	0
	Q3	0.4883	0.04	0.4438	0.6262	1	0.7819	0.8681	0.7659	0.7844	0.0073
	Max	0.92	0.73	1	0.98	1	1	1	1	1	0.77
acc	M	0.3441	0.0828	0.4035	0.1696	0.8521	0.5371	0.7476	0.5502	0.6187	0.2538
	SD	0.25011	0.13131	0.3316	0.1701	0.22326	0.3725	0.19165	0.36841	0.24033	0.32945
	Min	0	0	0	0	0	0	0.37	0	0.09	-0.23
	Q1	0	0	0.1642	0.004	0.6552	0.0161	0.6224	0	0.4852	0
	Q2	0.4938	0	0.3738	0.1319	1	0.7156	0.7539	0.765	0.6474	0.0015
	Q3	0.5012	0.1319	0.5738	0.2644	1	0.7894	0.8688	0.8823	0.6895	0.7369
	Max	0.92	0.39	1	0.62	1	1	1	1	1	0.77
F1	M	0.3531	0.1719	0.3144	0.1606	0.6815	0.6379	0.6675	0.5627	0.5457	0.2472
	SD	0.24004	0.19223	0.228	0.16351	0.3576	0.30946	0.18867	0.34467	0.22504	0.33727
	Min	0	0	0	0	0	0	0.24	0	0.07	-0.27
	Q1	0.0766	0	0.1537	0.0015	0.5316	0.4137	0.4995	0.2244	0.3816	-0.0009
	Q2	0.4938	0.0679	0.3564	0	0.8076	0.7567	0.6529	0.7643	0.6474	0.0468
	Q3	0.5012	0.3707	0.3807	0	1	0.8	0.8681	0.8823	0.6835	0.7368
	Max	0.92	0.76	1	0.62	1	1	1	0.96	1	0.77
*mcen**	M	0.0942	0.0527	0.4336	0.4194	0.9249	0.1629	0.5279	0.1279	0.8006	0.0091
	SD	0.22804	0.1313	0.31776	0.31942	0.20434	0.36455	0.27403	0.29611	0.10896	0.07837
	Min	0	0	0	0	0	0	0.02	0	0.42	-0.11
	Q1	0	0	0.2538	0.004	1	0	0.3543	0	0.7615	0
	Q2	0	0	0.3738	0.6262	1	0	0.3738	0	0.7672	0
	Q3	0	0	0.5901	0.6457	1	0	0.6456	0	0.8546	0
	Max	0.92	0.39	1	0.98	1	1	1	1	1	0.75
kappa	M	0.2961	0.1077	0.3786	0.2176	0.7942	0.5375	0.6747	0.4758	0.6525	0.2704
	SD	0.26073	0.18551	0.25057	0.22894	0.32691	0.38473	0.21607	0.39982	0.177	0.31679
	Min	0	0	0	0	0	0	0.02	0	0.18	-0.11
	Q1	0.0008	0	0.24	0.004	0.6999	0.0041	0.4995	0.0041	0.646	0
	Q2	0.4752	0.0007	0.3738	0.1319	0.999	0.7182	0.6949	0.7566	0.655	0.0958
	Q3	0.4943	0.1174	0	0.3095	1	0.7894	0.8681	0.8823	0.7301	0.7369
	Max	0.92	0.76	1	0.98	1	1	1	0.96	1	0.77

Table D.4: *Statistics about the 100 revised norms selected by different metrics in the case of alteration. The values in the table average the 100 independent revisions. Values for TP, FP, TN and FN are the percentage w.r.t. the total number of traces (i.e. 1500) in the dataset.*

metric		TP	FP	TN	FN	precision	recall	*acc*	*F1*	*mcen**	*kappa*
	M	0.2671	0.26	0.2263	0.2466	0.6199	0.5303	0.4934	0.378	0.6586	0.0833
	SD	0.29593	0.32215	0.2199	0.2779	0.39698	0.42853	0.27802	0.39086	0.23744	0.30645
	Min	0	0	0	0	0	0	0	0	0.03	-0.72
random	Q1	0	0	0.0002	0.0015	0.2135	0	0.3399	0	0.6328	-0.0027
	Q2	0.0833	0.0766	0.2412	0.1319	0.7099	0.7276	0.4862	0.2081	0.6854	0
	Q3	0.5394	0.3738	0.3738	0.5975	1	0.9704	0.7087	0.785	0.7668	0.074
	Max	0.98	1	1	0.98	1	1	1	0.99	1	0.77
	M	0.4374	0.1072	0.3792	0.0763	0.858	0.6702	0.8166	0.6414	0.6776	0.3041
	SD	0.28307	0.14685	0.35122	0.08277	0.2149	0.38168	0.13299	0.36872	0.19817	0.33154
	Min	0	0	0	0	0	0	0.55	0	0.16	-0.11
acc	Q1	0.0662	0	0.0228	0.0007	0.7244	0.6535	0.695	0.589	0.6429	0
	Q2	0.4943	0	0.3734	0.0486	1	0.7894	0.8681	0.8106	0.654	0.1846
	Q3	0.6248	0.2243	0.5114	0.1319	1	0.9966	0.8822	0.8823	0.6965	0.7369
	Max	0.99	0.4	1	0.41	1	1	1	1	1	0.77
	M	0.4665	0.2902	0.1962	0.0472	0.6415	0.904	0.6627	0.6598	0.6869	0.2798
	SD	0.2725	0.28965	0.19609	0.06063	0.37048	0.11874	0.24913	0.34117	0.1249	0.33673
	Min	0	0	0	0	0	0.33	0	0	0.24	-0.01
F1	Q1	0.3481	0.0007	0	0	0.4099	0.7896	0.5592	0.5815	0.6474	0
	Q2	0.495	0.2825	0.1142	0.002	0.7023	0.988	0.7016	0.8149	0.6753	0.0076
	Q3	0.6317	0.4348	0.3738	0.1306	0.9987	1	0.8681	0.8823	0.6987	0.7368
	Max	0.99	1	0.52	0.15	1	1	1	1	1	0.77
	M	0.1222	0.2876	0.1987	0.3914	0.7123	0.4053	0.321	0.1531	0.8231	0.008
	SD	0.24839	0.40039	0.19016	0.34061	0.4005	0.48937	0.22127	0.29789	0.10373	0.06789
	Min	0	0	0	0	0	0	0	0	0.68	0
*mcen**	Q1	0	0	0	0	0.4099	0	0.1727	0	0.7668	0
	Q2	0	0	0.2334	0.6262	1	0	0.3651	0	0.7771	0
	Q3	0.0057	0.5901	0.3738	0.6457	1	1	0.3738	0.0113	0.9017	0
	Max	0.99	1	1	0.98	1	1	1	1	1	0.67
	M	0.3579	0.1689	0.3174	0.1557	0.7358	0.7125	0.6753	0.5786	0.6259	0.3382
	SD	0.25972	0.25346	0.21444	0.18561	0.3629	0.30471	0.23366	0.37738	0.19493	0.32041
	Min	0	0	0	0	0	0	0	0	0.16	0
kappa	Q1	0.0118	0	0.1479	0.0015	0.6262	0.7057	0.5408	0.0941	0.6047	0.0013
	Q2	0.4943	0.018	0.3663	0.1319	0.9479	0.7894	0.7169	0.7723	0.6474	0.3056
	Q3	0.5097	0.3656	0.379	0.1749	1	0.9185	0.8681	0.8823	0.6974	0.7369
	Max	0.99	1	1	0.98	1	1	1	1	1	0.77

Bibliography

[1] Norsys Software Corp. (2021). Netica 6.08 bayesian network software from norsys. http://www.norsys.com. Accessed: 2021-01-01.

[2] Evan Ackerman. Tesla Working Towards 90 Percent Autonomous Car Within Three Years. http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/tesla-working-towards-90-autonomous-car-within-three-years. Accessed: 2020-09-28.

[3] Thomas Ågotnes and Michael J. Wooldridge. Optimal social laws. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010*, pages 667–674, 2010.

[4] Thomas Ågotnes, Wiebe van der Hoek, and Michael Wooldridge. Normative system games. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007*, page 129, 2007.

[5] Stéphane Airiau, Sandip Sen, and Daniel Villatoro. Emergence of conventions through social learning - heterogeneous learners in complex networks. *Autonomous Agents and Multi-Agent Systems*, 28(5):779–804, 2014.

[6] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, 1985.

[7] Huib Aldewereld. Autonomy vs. conformity: An institutional perspective on norms and protocols. *The Knowledge Engineering Review*, 24(4):410–411, 2009.

[8] Natasha Alechina, Mehdi Dastani, and Brian Logan. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012*, pages 1057–1064, 2012.

[9] Natasha Alechina, Mehdi Dastani, and Brian Logan. Reasoning about normative update. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 20–26, 2013.

[10] Natasha Alechina, Mehdi Dastani, and Brian Logan. Norm approximation for imperfect monitors. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014*, pages 117–124, 2014.

[11] Natasha Alechina, Nils Bulling, Mehdi Dastani, and Brian Logan. Practical run-time norm enforcement with bounded lookahead. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015*, pages 443–451, 2015.

[12] Natasha Alechina, Brian Logan, and Mehdi Dastani. Modeling norm specification and verification in multiagent systems. *FLAP*, 5(2):457–490, 2018.

[13] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal modeling framework for self-contextualizable software. In *Proceedings of the 10th International Workshop on Enterprise, Business-Process and Information Systems Modeling, BPMDS 2009, and 14th International Conference, EMMSAD 2009*, pages 326–338, 2009.

[14] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15 (4):439–458, 2010.

[15] Raian Ali, Fabiano Dalpiaz, Paolo Giorgini, and Vítor Estêvão Silva Souza. Requirements evolution: From assumptions to reality. In *Proceedings of the 12th International Conference on Enterprise, Business-Process and Information Systems Modeling, BPMDS 2011, and 16th International Conference, EMMSAD 2011*, pages 372–382, 2011.

[16] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information & Software Technology*, 55(1):35–57, 2013.

[17] André Almeida, Nelly Bencomo, Thaís Vasconcelos Batista, Everton Cavalcante, and Francisco Dantas. Dynamic decision-making based on NFR for managing software variability and configuration selection. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC 2015*, pages 1376–1382, 2015.

[18] Dalal Alrajeh, Oliver Ray, Alessandra Russo, and Sebastián Uchitel. Extracting requirements from scenarios with ILP. In *Proceedings of the 16th International Conference on Inductive Logic Programming, ILP 2006, Revised Selected Papers*, pages 64–78, 2006.

[19] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.

[20] Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leon van der Torre. Normative multi-agent systems (dagstuhl seminar 12111). *Dagstuhl Reports*, 2 (3):23–49, 2012.

[21] Annie I Anton. Goal-based requirements analysis. In *Proceedings of the 2nd IEEE International Conference on Requirements Engineering, RE 1996*, pages 136–144, 1996.

[22] Alexander Artikis and Jeremy Pitt. A formal model of open agent societies. In *Proceedings of the Fifth International Conference on Autonomous Agents, AGENTS 2001*, pages 192–193, 2001.

[23] Alexander Artikis and Marek J. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18(1):31–65, 2010.

[24] Duangtida Athakravi, Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Handling change in normative specifications. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012*, pages 1369–1370, 2012.

[25] Guillaume Aucher, Davide Grossi, Andreas Herzig, and Emiliano Lorini. Dynamic context logic. In *Proceedings of the Second International Workshop on Logic, Rationality, and Interaction, LORI 2009*, pages 15–26, 2009.

[26] Robert Axelrod. An evolutionary approach to norms. *The American political science review*, pages 1095–1111, 1986.

[27] Tim Baarslag, Michael Kaisers, Enrico H. Gerding, Catholijn M. Jonker, and Jonathan Gratch. When will negotiation agents be able to represent us? the challenges and opportunities for autonomous negotiators. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 4684–4690.

[28] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):5–27, 1998.

[29] Jorge A. Baier and Sheila A. McIlraith. On domain-independent heuristics for planning with qualitative preferences. In *Logical Formalizations of Commonsense Reasoning, Papers from the 2007 AAAI Spring Symposium, Technical Report SS-07-05*, pages 7–12, 2007.

[30] Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. A commitment-based infrastructure for programming socio-technical systems. *ACM Transactions on Internet Technology (TOIT)*, 14(4):1–23, 2014.

[31] Tina Balke, Célia da Costa Pereira, Frank Dignum, Emiliano Lorini, Antonino Rotolo, Wamberto Weber Vasconcelos, and Serena Villata. Norms in MAS: definitions and related concepts. In *Normative Multi-Agent Systems*, pages 1–31. 2013.

[32] Lina Barakat, Simon Miles, and Michael Luck. Adaptive composition in dynamic service environments. *Future Generation Computer Systems*, 80:215–228, 2018.

[33] Luciano Baresi and Carlo Ghezzi. The disappearing boundary between development-time and run-time. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 17–22, 2010.

[34] David A. Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zalinescu. Enforceable security policies revisited. *ACM Transactions on Information and System Security*, 16(1):3:1–3:26, 2013.

[35] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14:1–14:64, 2011.

[36] Gordon Baxter and Ian Sommerville. Socio-technical systems: From design methods to systems engineering. *Interacting with computers*, 23(1):4–17, 2011.

[37] Laszlo A. Belady and M. M. Lehman. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

[38] Nawal Benabbou and Patrice Perny. Adaptive elicitation of preferences under uncertainty in sequential decision making problems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 4566–4572, 2017.

[39] Nelly Bencomo. Quantun: Quantification of uncertainty for the reassessment of requirements. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference, RE 2015*, pages 236–240, 2015.

[40] Nelly Bencomo, Jon Whittle, Peter Sawyer, Anthony Finkelstein, and Emmanuel Letier. Requirements reflection: requirements as runtime entities. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010*, pages 199–202, 2010.

[41] Nelly Bencomo, Amel Belaggoun, and Valérie Issarny. Dynamic decision networks for decision-making in self-adaptive systems: a case study. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2013*, pages 113–122, 2013.

[42] Nelly Bencomo, Amel Belaggoun, and Valérie Issarny. Bayesian artificial intelligence for tackling uncertainty in self-adaptive systems: The case of dynamic decision networks. In *Proceedings of the 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2013*, pages 7–13, 2013.

[43] Nelly Bencomo, Robert B France, Betty HC Cheng, and Uwe Aßmann. *Models@ run. time: foundations, applications, and roadmaps*, volume 8378. Springer, 2014.

[44] Amel Bennaceur, Andrea Zisman, Ciaran McCormick, Danny Barthaud, and Bashar Nuseibeh. Won't take no for an answer: resource-driven requirements

adaptation. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019*, pages 77–88, 2019.

[45] Cristina Bicchieri. *The grammar of society: The nature and dynamics of social norms*. Cambridge University Press, 2005.

[46] Laura Bieker, Daniel Krajzewicz, AntonioPio Morra, Carlo Michelacci, and Fabio Cartolano. Traffic simulation for all: a real world traffic scenario from the city of bologna. In *Modeling Mobility with Open Data*, pages 47–60. Springer, 2015.

[47] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Planning with qualitative temporal preferences. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2006*, pages 134–144, 2006.

[48] Gordon Blair, Nelly Bencomo, and Robert B France. Models@ run. time. *Computer*, 42(10), 2009.

[49] Guido Boella and Leendert W. N. van der Torre. Regulative and constitutive norms in normative multiagent systems. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning, KR 2004*, pages 255–266, 2004.

[50] Guido Boella, Leendert Van Der Torre, and Harko Verhagen. Introduction to normative multiagent systems. *Computational & Mathematical Organization Theory*, 12(2-3):71–79, 2006.

[51] Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen. Introduction to normative multiagent systems. In *Normative Multi-agent Systems, 18.03. - 23.03.2007*, volume 07122 of *Dagstuhl Seminar Proceedings*, 2007.

[52] Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen. Introduction to the special issue on normative multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 17(1):1–10, 2008.

[53] Guido Boella, Gabriella Pigozzi, and Leendert W. N. van der Torre. Normative framework for normative system change. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009, Volume 1*, pages 169–176, 2009.

[54] Olivier Boissier, Rafael H. Bordini, Jomi Fred Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761, 2013.

[55] Kenneth Boness, Anthony Finkelstein, and Rachel Harrison. A lightweight technique for assessing risks in requirements analysis. *IET Software*, 2(1):46–57, 2008.

[56] Kenneth Boness, Anthony Finkelstein, and Rachel Harrison. A method for assessing confidence in requirements analysis. *Information & Software Technology*, 53(10):1084–1096, 2011.

[57] Rafael H. Bordini, Jomi Fred Hübner, and Renata Vieira. Jason and the golden fleece of agent-oriented programming. In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 3–37. 2005.

[58] Nick Bostrom and Eliezer Yudkowsky. The ethics of artificial intelligence. *The Cambridge handbook of artificial intelligence*, 1:316–334, 2014.

[59] Eva Bou, Maite López-Sánchez, and Juan A. Rodríguez-Aguilar. Adaptation of autonomic electronic institutions through norms and institutional agents. In *Proceedings of the 7th International Workshop on Engineering Societies in the Agents World, ESAW 2006, Revised Selected and Invited Papers*, pages 300–319, 2006.

[60] Pierre Bourque and Richard E. Fairley, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition. ISBN 978-0-7695-5166-1.

[61] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artificial Intelligence*, 170(8-9):686–713, 2006.

[62] Michael Bratman et al. *Intention, plans, and practical reason*, volume 10. Harvard University Press Cambridge, MA, 1987.

[63] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computer Intelligence*, 4:349–355, 1988.

[64] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[65] Jan M. Broersen, Mehdi Dastani, Joris Hulstijn, Zhisheng Huang, and Leendert W. N. van der Torre. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the Fifth International Conference on Autonomous Agents, AGENTS 2001*, pages 9–16, 2001.

[66] Jan M. Broersen, Mehdi Dastani, and Leendert W. N. van der Torre. Resolving conflicts between beliefs, obligations, intentions, and desires. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2001*, pages 568–579, 2001.

[67] Jan M. Broersen, Mehdi Dastani, and Leendert W. N. van der Torre. Beliefs, obligations, intentions, and desires as components in an agent architecture. *International Journal of Intelligent Systems*, 20(9):893–919, 2005.

[68] Nils Bulling and Mehdi Dastani. Norm-based mechanism design. *Artificial Intelligence*, 239:97–142, 2016.

[69] Nils Bulling, Mehdi Dastani, and Max Knobbout. Monitoring norm violations in multi-agent systems. In *Proceedings of the 12th International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2013*, pages 491–498, 2013.

[70] Birgit Burmeister, Afsaneh Haddadi, and Guido Matylis. Application of multi-agent systems in traffic and transportation. *Software Engineering - IEE Proceedings*, 144(1):51–60, 1997.

[71] Antoine Cailliau and Axel van Lamsweerde. Assessing requirements-related risks through probabilistic goals and obstacles. *Requirements Engineering*, 18 (2):129–146, 2013.

[72] Henrique Lopes Cardoso and Eugénio C. Oliveira. Adaptive deterrence sanctions in a normative framework. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009*, pages 36–43, 2009.

[73] Luís Carvalho. Smart cities from scratch? a socio-technical perspective. *Cambridge Journal of Regions, Economy and Society*, 8(1):43–60, 2015.

[74] Cristiano Castelfranchi. Prescribed mental attitudes in goal-adoption and norm-adoption. *Artificial Intelligence and Law*, 7(1):37–50, 1999.

[75] Cristiano Castelfranchi, Frank Dignum, Catholijn M. Jonker, and Jan Treur. Deliberative normative agents: Principles and architecture. In *Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages, ATAL 1999*, pages 364–378, 1999.

[76] Hei Chan and Adnan Darwiche. Sensitivity analysis in bayesian networks: From single to multiple parameters. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, UAI 2004*, pages 67–75, 2004.

[77] Li Chen and Pearl Pu. Survey of preference elicitation methods. Technical report, 2004.

[78] Amit Chopra, Leendert van der Torre, Harko Verhagen, and Serena Villata. *Handbook of Normative Multi-agent Systems, Draft, April 20, 2018*. 2018. URL `https://materials.dagstuhl.de/files/18/18171/18171.SWM.Preprint1.pdf`.

[79] Amit K. Chopra and Munindar P. Singh. From social machines to social protocols: Software engineering foundations for sociotechnical systems. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016*, pages 903–914, 2016.

[80] Amit K. Chopra, Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Reasoning about agents and protocols via goals and commitments. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010*, pages 457–464, 2010.

[81] Amit K Chopra, Fabiano Dalpiaz, F Başak Aydemir, Paolo Giorgini, John Mylopoulos, and Munindar P Singh. Protos: Foundations for engineering innovative sociotechnical systems. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference*, pages 53–62, 2014.

[82] George Christelis and Michael Rovatsos. Automated norm synthesis in an agent-based planning environment. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009, Volume 1*, pages 161–168, 2009.

[83] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Workshop on Logics of Programs*, pages 52–71, 1981.

[84] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[85] Lara Codeca and Jérôme Härri. Towards multimodal mobility simulation of C-ITS: the monaco SUMO traffic scenario. In *Proceedings of the 2017 IEEE Vehicular Networking Conference, VNC 2017*, pages 97–100, 2017.

[86] Lara Codeca and Jérôme Härri. Monaco sumo traffic (most) scenario: A 3d mobility scenario for cooperative its. *EPiC Series in Engineering*, 2:43–55, 2018.

[87] Barry H Cohen. *Explaining psychological statistics*. John Wiley & Sons, 2008.

[88] Don Cohen, Martin S. Feather, Khaled Narayanaswamy, and Stephen Fickas. Automatic monitoring of software requirements. In *Proceedings of the 19th International Conference on Software Engineering, ICSE 1997.*, pages 602–603, 1997.

[89] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[90] Jacob Cohen. Weighted kappa: nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, 70(4):213, 1968.

[91] James S Coleman. *Foundations of social theory*. Harvard university press, 1994.

[92] Giulianella Coletti, Davide Petturiti, and Barbara Vantaggi. Rationality principles for preferences on belief functions. *Kybernetika*, 51(3):486–507, 2015.

[93] Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. Autonomous norm acceptance. In *Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages, ATAL 1998*, pages 99–112, 1998.

[94] Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Normative design using inductive learning. *Theory and Practice of Logic Programming*, 11(4-5):783–799, 2011.

[95] Stephen Cranefield, Felipe Meneguzzi, Nir Oren, and Bastin Tony Roy Savarimuthu. A bayesian approach to norm identification. In *Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI 2016, Including Prestigious Applications of Artificial Intelligence, PAIS 2016*, pages 622–629, 2016.

[96] Elisabeth Crawford and Manuela M. Veloso. Learning dynamic preferences in multi-agent meeting scheduling. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2005*, pages 487–490.

[97] Thorsten Cziharz, Peter Hruschka, Stefan Queins, and Thorsten Weyer. Handbook of requirements modeling according to the IREB standard. Handbook, International Requirements Engineering Board, 2016.

[98] Fabiano Dalpiaz, Amit K. Chopra, Paolo Giorgini, and John Mylopoulos. Adaptation in open systems: Giving interaction its rightful place. In *Proceedings of the 29th International Conference on Conceptual Modeling*, pages 31–45, 2010.

[99] Fabiano Dalpiaz, Alexander Borgida, Jennifer Horkoff, and John Mylopoulos. Runtime goal models: Keynote. In *Proceedings of the 7th IEEE International Conference on Research Challenges in Information Science, RCIS 2013*, pages 1–11, 2013.

[100] Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering*, 18(1):1–24, 2013.

[101] Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. iStar 2.0 language guide. arXiv:1605.07767 [cs.SE], 2016.

[102] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1-2):3–50, 1993.

[103] Aniruddha Dasgupta and Aditya K. Ghose. BDI agents with objectives and preferences. In *Proceedings of the 8th International Workshop on Declarative Agent Languages and Technologies, DALT 2010, Revised, Selected and Invited Papers*, pages 22–39, 2010.

[104] Mehdi Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[105] Mehdi Dastani and Bas Testerink. Design patterns for multi-agent programming. *International Journal of Agent-Oriented Software Engineering*, 5(2/3): 167–202, 2016.

[106] Mehdi Dastani and Leendert W. N. van der Torre. Programming boid-plan agents: Deliberating about conflicts among defeasible mental attitudes and plans. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004*, pages 706–713, 2004.

[107] Mehdi Dastani, Frank Dignum, and John-Jules Ch. Meyer. Autonomy and agent deliberation. In *Postproceedings of the 1st International Workshop on Agents and Computational Autonomy - Potential, Risks, Solutions, AUTONOMY@AAMAS*, volume 2969, pages 114–127.

[108] Mehdi Dastani, Joris Hulstijn, and Leendert W. N. van der Torre. How to decide what to do? *European Journal of Operational Research*, 160(3):762–784, 2005.

[109] Mehdi Dastani, Davide Grossi, John-Jules Ch. Meyer, and Nick A. M. Tinnemeier. Normative multi-agent programs and their logics. In *Normative Multi-Agent Systems*, 2009.

[110] Mehdi Dastani, Sebastian Sardiña, and Vahid Yazdanpanah. Norm enforcement as supervisory control. In *Proceedings of the 20th International Conference on Principles and Practice of Multi-Agent Systems, PRIMA 2017*, pages 330–348, 2017.

[111] Paul Davidsson. Categories of artificial societies. In *Proceedings of the Second International Workshop on Engineering Societies in the Agents World, ESAW 2001, Revised Papers*, pages 1–9, 2001.

[112] Frank S de Boer, Koen V Hindriks, Wiebe van der Hoek, and J-J Ch Meyer. A verification framework for agent programming with declarative goals. *Journal of Applied Logic*, 5(2):277–302, 2007.

[113] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley R. Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ronald J. Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovski, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Richard D. Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II - International Seminar, 2010 Revised Selected and Invited Papers*, pages 1–32, 2010.

[114] Ana Cristina Vieira de Melo and Adilson de J. Sanchez. Software maintenance project delays prediction using bayesian networks. *Expert Systems with Applications*, 34(2):908–919, 2008.

[115] Isabel María del Águila and José del Sagrado. Bayesian networks for enhancement of requirements engineering: a literature review. *Requirements Engineering*, 21(4):461–480, 2016.

[116] Rosario Delgado and J David Núñez-González. Enhancing confusion entropy (cen) for binary and multiclass classification. *PloS one*, 14(1):e0210264, 2019.

[117] Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. Runtime norm revision using bayesian networks. In *Proceedings of the 21st International Conference on Principles and Practice of Multi-Agent Systems, PRIMA 2018*.

[118] Davide Dell'Anna, Fabiano Dalpiaz, and Mehdi Dastani. Validating goal models via bayesian networks. In *Proceedings of the 5th International Workshop on Artificial Intelligence for Requirements Engineering, AIRE@RE 2018*, pages 39–46. IEEE, 2018.

[119] Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. Runtime revision of norms and sanctions based on agent preferences. In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2019*, pages 1609–1617, 2019.

[120] Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. Runtime Revision of Sanctions in Normative Multi-Agent Systems - Supplementary Material - SASS, 2020.

[121] Davide Dell'Anna, Mehdi Dastani, and Fabiano Dalpiaz. Runtime revision of sanctions in normative multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 34(2):1–54, 2020.

[122] Daniel Clement Dennett. *The intentional stance*. MIT press, 1989.

[123] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011, Volume 1-3*, pages 225–232, 2011.

[124] Frank Dignum and Ruurd Kuiper. Combining dynamic deontic logic and temporal logic for the specification of deadlines. In *Proceedings of the 30th Annual Hawaii International Conference on System Sciences, HICSS-30*, pages 336–346, 1997.

[125] Frank Dignum, David N. Morley, Liz Sonenberg, and Lawrence Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on Multi-Agent Systems, ICMAS 2000*, pages 111–118, 2000.

[126] Frank Dignum, Virginia Dignum, and Catholijn M. Jonker. Towards agents for policy making. In *Proceedings of the Ninth International Workshop on Multi-Agent-Based Simulation, MABS 2008, Revised Selected Papers*, pages 141–153, 2008.

[127] Ozge Doguc and Jose Emmanuel Ramirez-Marquez. A generic method for estimating system reliability using bayesian networks. *Reliability Engineering System Safety*, 94(2):542–550, 2009.

[128] Sašo Dzeroski and Nada Lavrac. Inductive logic programming: Techniques and applications, 1994.

[129] Sašo Džeroski and Nada Lavrač. An introduction to inductive logic programming. In *Relational data mining*, pages 48–73. Springer, 2001.

[130] Jon Elster. *The cement of society: A survey of social order*. Cambridge university press, 1989.

[131] Joshua M Epstein. Learning to be thoughtless: Social norms and individual computation. *Computational economics*, 18(1):9–24, 2001.

[132] Clifton A Ericson and Clifton Ll. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, volume 1, pages 1–9, 1999.

[133] Neil A Ernst, John Mylopoulos, and Yiqiao Wang. Requirements evolution and what (research) to do about it. *Design Requirements Engineering: A Ten-Year Perspective*, 14:186–214, 2009.

[134] Neil A. Ernst, Alexander Borgida, Ivan Jureta, and John Mylopoulos. An overview of requirements evolution. In Mens et al. [205], pages 3–32.

[135] Marc Esteva, David de la Cruz, and Carles Sierra. ISLANDER: an electronic institutions editor. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002*, pages 1045–1052, 2002.

[136] Marc Esteva, Bruno Rosell, Juan A. Rodríguez-Aguilar, and Josep Lluís Arcos. AMELI: an agent-based middleware for electronic institutions. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS 2004*, pages 236–243, 2004.

[137] Owain Evans, Andreas Stuhlmüller, and Noah D. Goodman. Learning the preferences of ignorant, inconsistent agents. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 323–329. AAAI Press, 2016.

[138] Norman E. Fenton, Martin Neil, William Marsh, Peter Stewart Hearty, David Marquez, Paul Krause, and Rajat Mishra. Predicting software defects in varying development lifecycles using bayesian nets. *Information & Software Technology*, 49(1):32–43, 2007.

[139] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *Proceedings of the 4th International Workshop on Agent-Oriented Software Engineering, AOSE 2003, Revised Papers*.

[140] Stephen Fickas and Martin S. Feather. Requirements monitoring in dynamic environments. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering, RE 1995*, pages 140–147, 1995.

[141] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, 24(2):163–186, 2012.

[142] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas B. Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Amir Molzam Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. Software engineering meets control theory. In *Proceedings of the 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 71–82, 2015.

[143] Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 42(1):75–99, 2016. doi: 10.1109/TSE.2015.2421318.

[144] Anthony Finkelstein and Andrea Savigni. A framework for requirements engineering for context-aware services. IEEE Computer Society Press, 2001.

[145] Peter C Fishburn. Axioms for lexicographic preferences. *The Review of Economic Studies*, 42(3):415–419, 1975.

[146] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[147] Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe'er. Using bayesian networks to analyze expression data. *Journal of computational biology*, 7(3-4): 601–620, 2000.

[148] Peter Gärdenfors. *Belief revision*, volume 29. Cambridge University Press, 2003.

[149] Ricardo F Garzia, Mario R Garzia, and Bernard P Zeigler. Discrete-event simulation: When prototyping of large, complex systems is impossible, this technique, based on known behaviors of interacting elements, is useful. *IEEE Spectrum*, 23(12):32–36, 1986.

[150] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence, AAAI 1987*, pages 677–682, 1987.

[151] Sepideh Ghanavati, André Rifaut, Eric Dubois, and Daniel Amyot. Goal-oriented compliance with multiple regulations. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference, RE 2014*, pages 73–82, 2014.

[152] Eric Ghysels and Massimiliano Marcellino. *Applied economic forecasting using time series methods*. Oxford University Press, 2018.

[153] Jack P Gibbs. Norms: The problem of definition and classification. *American Journal of Sociology*, 70(5):586–594, 1965.

[154] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Reasoning with goal models. In *Proceedings of the 21st International Conference on Conceptual Modeling, ER 2002*, pages 167–181, 2002.

[155] Martin Glinz. On non-functional requirements. In *Proceedings of the 15th IEEE International Requirements Engineering Conference, RE 2007*, pages 21–26, 2007.

[156] Guido Governatori and Antonino Rotolo. Changing legal systems: legal abrogations and annulments in defeasible logic. *Logic Journal of the IGPL*, 18(1): 157–194, 2010.

[157] Danny Greefhorst and Erik Proper. *Architecture Principles - The Cornerstones of Enterprise Architecture*, volume 4 of *The Enterprise Engineering Series*. Springer, 2011.

[158] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[159] Susan Harker, Ken D. Eason, and John E. Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *Proceedings of IEEE International Symposium on Requirements Engineering, RE 1993*, pages 266–272, 1993.

[160] Daniel M Hausman. *Preference, value, choice, and welfare*. Cambridge University Press, 2011.

[161] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002*, pages 342–356, 2002.

[162] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[163] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. MOISE+: towards a structural, functional, and deontic model for MAS organization. In *Proceedings of The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002*, pages 501–502, 2002.

[164] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. S-moise$^+$: A middleware for developing organised multi-agent systems. In *Proceedings of the AAMAS 2005 International Workshops on Coordination, Organizations,*

Institutions, and Norms in Multi-Agent Systems, Agents, Norms and Institutions for Regulated Multi-Agent Systems, ANIREM 2005, and Organizations in Multi-Agent Systems, OOOP 2005, Revised Selected Papers, pages 64–78, 2005.

[165] ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Standard, International Organization for Standardization, March 2011.

[166] Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[167] Jie Jiang, Huib Aldewereld, Virginia Dignum, and Yao-Hua Tan. Norm contextualization. In *Proceedings of the 14th International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems, COIN@AAMAS 2012, Revised Selected Papers*, pages 141–157, 2012.

[168] Jie Jiang, John Thangarajah, Huib Aldewereld, and Virginia Dignum. Reasoning with agent preferences in normative multi-agent systems. In *Proceedings of the 13TH International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2014*, pages 1373–1374, 2014.

[169] Andrew JI Jones, Marek Sergot, et al. On the characterisation of law and computer systems: The normative systems perspective. *Deontic logic in computer science: normative system specification*, pages 275–307, 1993.

[170] Ivan Jureta, John Mylopoulos, and Stéphane Faulkner. Revisiting the core ontology and problem in requirements engineering. In *Proceedings of the 16th IEEE International Requirements Engineering Conference, RE 2008*, pages 71–80, 2008.

[171] Sohag Kabir and Yiannis Papadopoulos. Applications of bayesian networks and petri nets in safety, reliability, and risk assessments: A review. *Safety science*, 115:154–175, 2019.

[172] Özgür Kafali, Nirav Ajmeri, and Munindar P. Singh. Kont: Computing tradeoffs in normative multiagent systems. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pages 3006–3012, 2017.

[173] Stig Kanger. New Foundations for ethical theory. Deontic Logic: Introductory and Systematic Readings. Ed. Hilpinen, R, 1971.

[174] Abhilash Kantamneni, Laura E Brown, Gordon Parker, and Wayne W Weaver. Survey of multi-agent systems for microgrid control. *Engineering applications of artificial intelligence*, 45:192–203, 2015.

[175] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[176] Dongsun Kim and Sooyong Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009*, pages 76–85, 2009.

[177] Alessia Knauss, Daniela Damian, Xavier Franch, Angela Rook, Hausi A. Müller, and Alex Thomo. Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime. *Information & Software Technology*, 70: 85–99, 2016.

[178] Max Knobbout and Mehdi Dastani. Reasoning under compliance assumptions in normative multiagent systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012*, pages 331–340, 2012.

[179] Max Knobbout, Mehdi Dastani, and John-Jules Ch. Meyer. Reasoning about dynamic normative systems. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence, JELIA 2014*, pages 628–636, 2014.

[180] Max Knobbout, Mehdi Dastani, and John-Jules Ch. Meyer. A dynamic logic of norm change. In *Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI 2016 - Including Prestigious Applications of Artificial Intelligence, PAIS 2016*, pages 886–894, 2016.

[181] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995*, pages 1137–1145, 1995.

[182] Martin J. Kollingbaum and Timothy J. Norman. Noa - A normative agent architecture. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI 2003*, pages 1465–1466, 2003.

[183] Martin J. Kollingbaum and Timothy J. Norman. Norm adoption and consistency in the noa agent architecture. In *Proceedings of the First International Workshop on Programming Multi-Agent Systems, PROMAS 2003, Selected Revised and Invited Papers*, pages 169–186, 2003.

[184] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.

[185] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.

[186] Johan Kwisthout. Most probable explanations in bayesian networks: Complexity and tractability. *International Journal of Approximate Reasoning*, 52(9): 1452–1469, 2011.

[187] Kurt Lautenbach, Stephan Philippi, and Alexander Pinl. Bayesian networks and petri nets. *Entwurf komplexer Autimatisierungssysteme, EKA*, 9, 2006.

[188] Jeehang Lee, Julian A. Padget, Brian Logan, Daniela Dybalova, and Natasha Alechina. N-jason: Run-time norm compliance in agentspeak(l). In *Proceedings of the Second International Workshop on Engineering Multi-Agent Systems, EMAS@AAMAS 2014, Revised Selected Papers*, pages 367–387, 2014.

[189] Meir M Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[190] Meir M Lehman. The role and impact of assumptions in software development, maintenance and evolution. In *Proceedings of the 2005 IEEE International Workshop on Software Evolvability, Software-Evolvability 2005*, pages 3–14. IEEE, 2005.

[191] Meir M. Lehman and Juan F. Ramil. Software evolution - background, theory, practice. *Information Processing Letters*, 88(1-2):33–44, 2003.

[192] Emmanuel Letier and Axel van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, SIGSOFT 2004*, pages 53–62, 2004.

[193] Jack S Levy. Case studies: Types, designs, and logics of inference. *Conflict management and peace science*, 25(1):1–18, 2008.

[194] David Lewis. *Convention: A philosophical study.* John Wiley & Sons, 2008.

[195] Feng-Lin Li, Jennifer Horkoff, John Mylopoulos, Renata S. S. Guizzardi, Giancarlo Guizzardi, Alexander Borgida, and Lin Liu. Non-functional requirements as qualities, with a spice of ontology. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference, RE 2014*, pages 293–302, 2014.

[196] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1-2):2–16, 2005.

[197] Maite López-Sánchez, Marc Serramia, Juan A. Rodríguez-Aguilar, Javier Morales, and Michael Wooldridge. Automating decision making to help establish norm-based regulations. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pages 1613–1615, 2017.

[198] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.

[199] Cara MacNish and Mary-Anne Williams. From belief revision to design revision: Applying theory change to changing requirements. In *Proceedings of the PRICAI'96 Workshops on Reasoning with Incomplete and Changing Information and on Inducing Complex Representations, PRICAI 1996, Selected Papers*, pages 206–220, 1996.

[200] Moamin A Mahmoud, Mohd Sharifuddin Ahmad, Mohd Zaliman M Yusoff, and Salama A Mostafa. A regulative norms mining algorithm for complex adaptive system. In *International Conference on Soft Computing and Data Mining*, pages 213–224. Springer, 2018.

[201] David Makinson and Leendert Van Der Torre. Input/output logics. *Journal of philosophical logic*, 29(4):383–408, 2000.

[202] Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.

[203] C Masolo, S Borgo, A Gangemi, N Guarino, A Oltramari, and L Schneider. Dolce: a descriptive ontology for linguistic and cognitive engineering. *WonderWeb Project, Deliverable D17 v2*, 1:75–105, 2003.

[204] Emilia Mendes and Nile Mosley. Bayesian network models for web effort prediction: A comparative study. *IEEE Transactions on Software Engineering (TSE)*, 34(6):723–737, 2008.

[205] Tom Mens, Alexander Serebrenik, and Anthony Cleve, editors. *Evolving Software Systems*. Springer, 2014.

[206] John-Jules Ch Meyer, Roel J Wieringa, et al. Deontic logic in computer science normative system specification. 1993.

[207] Jordi Campos Miralles, Maite López-Sánchez, Juan A. Rodríguez-Aguilar, and Marc Esteva. Formalising situatedness and adaptation in electronic institutions. In *Coordination, Organizations, Institutions and Norms in Agent Systems IV, COIN 2008 International Workshops, COIN@AAMAS 2008, Estoril, Portugal, May 12, 2008. COIN@AAAI 2008, Chicago, USA, July 14, 2008. Revised Selected Papers*, pages 126–139, 2008.

[208] Jordi Campos Miralles, Maite López-Sánchez, Maria Salamó, Pedro Avila, and Juan A. Rodríguez-Aguilar. Robust regulation adaptation in multi-agent systems. *TAAS*, 8(3):13:1–13:27, 2013.

[209] Ayse Tosun Misirli and Ayse Basar Bener. A mapping study on bayesian networks for software quality prediction. In *3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2014*, pages 7–11, 2014.

[210] Javier Morales, Maite López-Sánchez, Juan A Rodríguez-Aguilar, Michael Wooldridge, and Wamberto Vasconcelos. Synthesising liberal normative systems. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015*, pages 433–441, 2015.

[211] Javier Morales, Maite López-Sánchez, Juan Antonio Rodríguez-Aguilar, Wamberto Weber Vasconcelos, and Michael J. Wooldridge. Online automated synthesis of compact normative systems. *ACM Transactions on Autonomous and Adaptive Systems*, 10(1):2:1–2:33, 2015.

[212] Javier Morales, Michael Wooldridge, Juan A. Rodríguez-Aguilar, and Maite López-Sánchez. Evolutionary synthesis of stable normative systems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pages 1646–1648, 2017.

[213] Javier Morales, Michael Wooldridge, Juan A Rodríguez-Aguilar, and Maite López-Sánchez. Synthesising evolutionarily stable normative systems. *arXiv preprint arXiv:1710.00709*, 2017.

[214] Mirko Morandini, Loris Penserini, and Anna Perini. Operational semantics of goal models in adaptive agents. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009, Volume 1*, pages 129–136, 2009.

[215] Salama A. Mostafa, Mohd Sharifuddin Ahmad, and Aida Mustapha. Adjustable autonomy: a systematic literature review. *Artificial Intelligence Review*, 51(2): 149–186, 2019.

[216] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, 18(6):483–497, 1992.

[217] Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning, ICML 2005*.

[218] Douglass C North. Institutions and a transaction-cost theory of exchange. *Perspectives on positive political economy*, 182:191, 1990.

[219] Robert Olszewski, Piotr Palka, and Agnieszka Turek. Solving "smart city" transport problems by designing carpooling gamification schemes with multi-agent systems: The case of the so-called "mordor of warsaw". *Sensors*, 18(1): 141, 2018.

[220] Shmuel Onn and Moshe Tennenholtz. Determination of social laws for multi-agent mobilization. *Artificial Intelligence*, 95(1):155–167, 1997.

[221] Elinor Ostrom. *Governing the commons: The evolution of institutions for collective action*. Cambridge university press, 1990.

[222] Elinor Ostrom. *Understanding institutional diversity*. Princeton university press, 2009.

[223] Lin Padgham and Dhirendra Singh. Situational preferences for BDI plans. In *Proceedings of the 12th International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2013*, pages 1013–1020, 2013.

[224] Laurence A. F. Park and Jesse Read. A blended metric for multi-label optimisation and evaluation. In *Proceedings of the 2018 European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2018*, pages 719–734, 2018.

[225] Tharindu Patikirikorala, Alan W. Colman, Jun Han, and Liuping Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012*, pages 33–42, 2012.

[226] Luis Hernán García Paucar, Nelly Bencomo, and Kevin Kam Fung Yuen. Juggling preferences in a world of uncertainty. In *Proceedings of the 25th IEEE International Requirements Engineering Conference, RE 2017*, pages 430–435, 2017.

[227] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[228] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In *Proceedings of the 14th International Symposium on Formal Methods, FM 2006*, pages 573–586, 2006.

[229] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering - Foundations, Principles, and Techniques.* Springer, 2005.

[230] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI reasoning engine. In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 149–174. 2005.

[231] Daniele Porello, Francesco Setti, Roberta Ferrario, and Marco Cristani. Multiagent socio-technical systems: An ontological approach. In *Proceedings of Ninth International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems, COIN@AAMAS 2013, Revised Selected Papers*, pages 42–62, 2013.

[232] Andres J. Ramirez, Erik M. Fredericks, Adam C. Jensen, and Betty H. C. Cheng. Automatically relaxing a goal model to cope with uncertainty. In *Proceedings of the 4th International Symposium on Search Based Software Engineering, SSBSE 2012*, pages 198–212, 2012.

[233] Anand S. Rao. Agentspeak(l): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW 1996*, pages 42–55, 1996.

[234] Sandeep Reddivari, Shirin Rad, Tanmay Bhowmik, Nisreen Cain, and Nan Niu. Visual requirements analytics: a framework and case study. *Requirements Engineering*, 19(3):257–279, 2014.

[235] Rijkswaterstaat. De maximumsnelheid in Nederland. `https://www.rijkswaterstaat.nl/wegen/wetten-regels-en-vergunningen/verkeerswetten/maximumsnelheid/index.aspx`. Accessed: 2020-09-28 (in Dutch).

[236] William N. Robinson. A requirements monitoring framework for enterprise systems. *Requirements Engineering*, 11(1):17–41, 2006.

[237] Mariacristina Roscia, Michela Longo, and George Cristian Lazaroiu. Smart city by multi-agent systems. In *Proceedings of the 2013 International Conference on Renewable Energy Research and Applications, ICRERA 2013*, pages 371–376, 2013.

[238] Michael Rosemann, Jan Recker, and Christian Flender. Contextualisation of business processes. *IJBPIM*, 3(1):47–60, 2008.

[239] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, 1994.

[240] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. ISBN 978-0-13-207148-2.

[241] Mazeiar Salehie and Ladan Tahvildari. Towards a goal-driven approach to action selection in self-adaptive software. *Software - Practice and Experience*, 42(2): 211–233, 2012.

[242] Bastin Tony Roy Savarimuthu and Stephen Cranefield. Norm creation, spreading and emergence: A survey of simulation models of norms in multi-agent systems. *Multiagent and Grid Systems*, 7(1):21–54, 2011.

[243] Peter Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-aware systems: A research agenda for RE for self-adaptive systems. In *Proceedings of the 18th IEEE International Requirements Engineering Conference, RE 2010*, pages 95–103, 2010.

[244] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2-3):135–168, 2000.

[245] Sanny Schmid, Ilias Gerostathopoulos, Christian Prehofer, and Tomás Bures. Self-adaptation based on big data analytics: A model problem and tool. In *Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2017*, pages 102–108, 2017.

[246] Marco Scutari. Learning bayesian networks with the bnlearn r package. *arXiv preprint arXiv:0908.3817*, 2009.

[247] John R Searle, S Willis, et al. *The construction of social reality*. Simon and Schuster, 1995.

[248] Estefanía Serral, Paolo Sernani, Aldo Franco Dragoni, and Fabiano Dalpiaz. Contextual requirements prioritization and its application to smart homes. In Andreas Braun, Reiner Wichert, and Antonio Maña, editors, *Proceedings of the 13th European Conference on Ambient Intelligence, AmI 2017*.

[249] Estefanía Serral, Paolo Sernani, and Fabiano Dalpiaz. Personalized adaptation in pervasive systems via non-functional requirements. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2017.

[250] Glenn Shafer. Dempster-shafer theory. *Encyclopedia of artificial intelligence*, 1: 330–331, 1992.

[251] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[252] Yan Shen, Jianwen Li, Zheng Wang, Ting Su, Bin Fang, Geguang Pu, Wanwei Liu, and Mingsong Chen. Runtime verification by convergent formula progression. In *Proceedings of the 21st Asia-Pacific Software Engineering Conference, APSEC 2014, Volume 1: Research Papers*, pages 255–262, 2014.

[253] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[254] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI 1992.*, pages 276–281, 1992.

[255] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial intelligence*, 73(1-2):231–252, 1995.

[256] Munindar P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Transactions on Intelligent Systems and Technology*, 5(1):21:1–21:23, 2013.

[257] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems, NIPS 2012*, pages 2960–2968, 2012.

[258] Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Z. Kwiatkowska, John A. McDermid, and Richard F. Paige. Large-scale complex IT systems. *Communications of the ACM*, 55(7):71–77, 2012.

[259] Mohammad S Sorower. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18:1–25, 2010.

[260] Vítor Estêvão Silva Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness requirements for adaptive systems. In *Proceedings of the 2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011*, pages 60–69, 2011.

[261] David J Spiegelhalter, A Philip Dawid, Steffen L Lauritzen, and Robert G Cowell. Bayesian analysis in expert systems. *Statistical science*, pages 219–247, 1993.

[262] Yuki Sugiyama, Minoru Fukui, Macoto Kikuchi, Katsuya Hasebe, Akihiro Nakayama, Katsuhiro Nishinari, Shin-ichi Tadaki, and Satoshi Yukawa. Traffic jams without bottlenecks—experimental evidence for the physical mechanism of the formation of a jam. *New journal of physics*, 10(3):033001, 2008.

[263] Alistair Sutcliffe and Pete Sawyer. Requirements elicitation: Towards the unknown unknowns. In *Proceedings of the 21st IEEE International Requirements Engineering Conference, RE 2013*, pages 92–104, 2013.

[264] Alistair G Sutcliffe. Requirements analysis for socio-technical system design. *Information Systems*, 25(3):213–233, 2000.

[265] Bas Testerink, Mehdi Dastani, and Nils Bulling. Distributed controllers for norm enforcement. In *Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI 2016 - Including Prestigious Applications of Artificial Intelligence PAIS 2016*, pages 751–759, 2016.

[266] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2018.

[267] Nick A. M. Tinnemeier, Mehdi Dastani, John-Jules Ch. Meyer, and Leendert W. N. van der Torre. Programming normative artifacts with declarative obligations and prohibitions. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009*, pages 145–152, 2009.

[268] Maksim Tsvetovat and Kathleen M Carley. Modeling complex socio-technical systems using multi-agent simulation methods. *KI*, 18(2):23–28, 2004.

[269] Kagan Tumer, Zachary T. Welch, and Adrian K. Agogino. Aligning social welfare and agent preferences to alleviate traffic congestion. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2008), Volume 2*, pages 655–662, 2008.

[270] Raimo Tuomela and Maj Bonnevier-Tuomela. Norms and agreement. *Social Ontology in the Making*, page 283, 2020.

[271] Ed van Akkeren, Lars Baumann, Jan Jaap Cannegieter, Colin Hood, Peter Hruschka, Matthias Lampe, Ellen Leutbecher, Hans van Loenhoud, Piet de Roo, Stefan Staal, et al. Handbook of requirements modeling according to the ireb standard. *IREB International Requirements Engineering Board eV*, 2016.

[272] Johan van Benthem and Fenrong Liu. Dynamic logic of preference upgrade. *Journal of Applied Non-Classical Logics*, 17(2):157–182, 2007.

[273] Wil Van Der Aalst, Arya Adriansyah, and Boudewijn Van Dongen. Causal nets: a modeling language tailored towards process discovery. In *International conference on concurrency theory*, pages 28–42. Springer, 2011.

[274] Linda van der Gaag, Silja Renooij, and Veerle Coupé. Sensitivity analysis of probabilistic networks. *Advances in probabilistic graphical models*, pages 103–124, 2007.

[275] Axel Van Lamsweerde. *Requirements engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons, 2009.

[276] Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering (TSE)*, 24(11):908–926, 1998.

[277] M. Birna van Riemsdijk, Mehdi Dastani, and John-Jules Ch. Meyer. Subgoal semantics in agent programming. In *Proceedings of the 12th Portuguese Conference on Artificial Intelligence, EPIA 2005*, pages 548–559, 2005.

[278] M. Birna van Riemsdijk, Mehdi Dastani, and Michael Winikoff. Goals in agent systems: a unifying framework. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2008, Volume 2*, pages 713–720, 2008.

[279] M. Birna van Riemsdijk, Koen V. Hindriks, and Catholijn M. Jonker. Programming organization-aware agents. In *Proceedings of the 10th International Workshop on Engineering Societies in the Agents World, ESAW 2009*, pages 98–112, 2009.

[280] Pravin Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on automatic control*, 38(2):195–207, 1993.

[281] Wamberto Weber Vasconcelos, Martin J. Kollingbaum, and Timothy J. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19(2):124–152, 2009.

[282] Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum. Implementing norms in multiagent systems. In *Proceedings of the Second German Conference on Multiagent System Technologies, MATES 2004*, pages 313–327, 2004.

[283] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.

[284] Javier Vázquez-Salceda, Huib Aldewereld, Davide Grossi, and Frank Dignum. From human regulations to regulated software agents' behavior. *Artificial Intelligence and Law*, 16(1):73–87, 2008.

[285] Thiago Viana, Andrea Zisman, and Arosha K. Bandara. Identifying conflicting requirements in systems of systems. In *25th IEEE International Requirements Engineering Conference, RE 2017*, pages 436–441, 2017.

[286] Simeon Visser, John Thangarajah, James Harland, and Frank Dignum. Preference-based reasoning in BDI agent systems. *Autonomous Agents and Multi-Agent Systems*, 30(2):291–330, 2016.

[287] Georg Henrik Von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

[288] Haiqin Wang, Irina Rish, and Sheng Ma. Using sensitivity analysis for selective parameter update in bayesian network learning. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2002.

[289] Yiqiao Wang and John Mylopoulos. Self-repair through reconfiguration: A requirements engineering approach. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering, ASE 2009*, pages 257–268, 2009.

[290] Yiqiao Wang, Sheila A. McIlraith, Yijun Yu, and John Mylopoulos. Monitoring and diagnosing software requirements. *Automated Software Engineering*, 16(1): 3–35, 2009.

[291] Jin-Mao Wei, Xiao-Jie Yuan, Qing-Hua Hu, and Shu-Qin Wang. A novel measure for evaluating classifiers. *Expert Systems with Applications*, 37(5):3799–3809, 2010.

[292] Michael P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44(3):257–303, 1990.

[293] Michael P. Wellman and Max Henrion. Explaining 'explaining away'. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 15(3): 287–292, 1993.

[294] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196, 2010.

[295] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.

[296] Roel Wieringa and J-J Ch Meyer. Actors, actions, and initiative in normative system specification. *Annals of mathematics and artificial intelligence*, 7(1-4): 289–346, 1993.

[297] Michael J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)*. Wiley, 2009. ISBN 978-0-470-51946-2.

[298] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[299] Haochi Wu, Lin Liu, and Wenting Ma. Optimizing requirements elicitation with an i* and bayesian network integrated modelling approach. In *Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference Workshops, COMPSAC Workshops 2010*, pages 182–188, 2010.

[300] Paul Pao-Yen Wu, Clinton Fookes, Jegar Pitchforth, and Kerrie Mengersen. A framework for model integration and holistic modelling of socio-technical systems. *Decision Support Systems*, 71:14–27, 2015.

[301] Fabiola López y López, Michael Luck, and Mark d'Inverno. A normative framework for agent-based systems. *Computational Mathematical Organization Theory*, 12(2-3):227–250, 2006.

[302] Eric Yu. Modeling strategic relationships for process reengineering. *Social Modeling for Requirements Engineering*, 11(2011):66–87, 2011.

[303] Eric Yu and John Mylopoulos. Why goal-oriented requirements engineering. In *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, REFSQ 1998*, volume 15, pages 15–22, 1998.

[304] Franco Zambonelli, Nicholas R. Jennings, and Michael J. Wooldridge. Organizational abstractions for the analysis and design of multi-agent systems. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering, AOSE 2000, Revised Papers*, pages 235–251, 2000.

[305] Franco Zambonelli, Nicholas R. Jennings, and Michael J. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11 (3):303–328, 2001.

[306] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997.

[307] Lin Zhao, Tao Tang, Jinzhao Wu, and Tianhua Xu. Runtime verification with multi-valued formula rewriting. In *Proceedings of the 4th IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE 2010*, pages 77–86, 2010.

[308] Brian D. Ziebart, Anind K. Dey, and J. Andrew Bagnell. Fast planning for dynamic preferences. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling, ICAPS 2008*, pages 412–419. AAAI, 2008.

[309] Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information & Software Technology*, 45(14):993–1009, 2003.

[310] Didar Zowghi and Ray Offen. A logical framework for modeling and reasoning about the evolution of requirements. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, ISRE 1997*, pages 247–257. IEEE, 1997.

Summary

Modern software systems execute in increasingly dynamic settings, and their objectives are in constant motion. In order to preserve their adequacy and effectiveness within an evolving environment, software and its requirements need to adapt to change. In this dissertation, we propose a data-driven supervision framework for the automatic run-time revision of requirements, so to ensure the achievement of system-level objectives in dynamic settings.

We focus on the supervision of multi-agent systems (MASs), collections of interacting autonomous agents, such as autonomous cars on smart roads. In multi-agent systems, agents' internals are typically unknown to the other agents and to the MAS designer. Norms are often employed as a means for controlling and coordinating the agents' behavior without over-constraining their autonomy. We use norms to characterize requirements for the behavior of the agents in the system, and we use sanctions as a deterrence mechanism to discourage agents from violations.

The proposed supervision framework employs a general architecture for system self-adaptation, described as a closed control-loop. At run-time, the system is monitored and execution data is collected in different operating contexts. The collected data is used to learn statistical correlations between the achievement of the system's objectives and the satisfaction of the requirements in the different operating contexts. The learnt information is applied to automatically assess the validity of the assumptions made at design-time, and to automatically synthesise new requirements and sanctions when there is evidence that the current ones are not effective.

Samenvatting

Moderne softwaresystemen opereren in steeds dynamischere contexten met constant veranderende doelen. Om de effectiviteit te garanderen zouden zowel de eisen aan de software als de software zelf in runtime aangepast moeten kunnen worden. Dit proefschrift presenteert een raamwork voor datagestuurde supervisie van operationele softwaresystemen. Het supervisiesysteem past softwaresystemen in runtime aan en daarmee garandeert de realisatie van de systeemdoelen in dynamische contexten.

Wij richten ons op de supervisie van multi-agent systemen (MAS'en). De softwareagenten in multi-agent systemen, zoals zelfrijdende auto's op slimme wegen, zijn doorgaans onbekend voor andere softwareagenten en voor de ontwikkelteams van het MAS. Normen worden vaak ingezet om het gedrag van agenten te observeren en te coördineren zonder hun autonomie te beperken. Wij gebruiken normen om het gewenste gedrag van de softwareagenten te karakteriseren en gebruiken sancties als een mechanisme om softwareagenten te ontmoedigen deze normen te schenden.

Het supervisieraamwerk is gebaseerd op een generieke architectuur voor het aanpassen van het systeem. Het operationele systeem wordt gemonitord en de observatiedata van verschillende actieve contexten worden verzameld. Deze data wordt gebruikt om statistische correlaties tussen het bereiken van de systeemdoelen en de mate waarin aan de eisen wordt voldaan in verschillende actieve contexten te leren. Deze berekeningen maken het mogelijk om de validiteit van de ontwerpaannames te testen en automatisch nieuwe eisen en sancties te genereren wanneer de huidige eisen en sancties niet effectief lijken te zijn.

Acknowledgements

I would like to thank Mehdi Dastani and Fabiano Dalpiaz. It was a privilege to have you as supervisors and I am extremely grateful for your guidance and support during these years. Thank you for giving me the possibility to pursue my ideas autonomously, and for the many hours you dedicated to help me developing, improving and formalizing those ideas. I have learned a lot from both of you and from our discussions. Working with you has been an honour and an immense source of growth and inspiration, not only academically but also personally.

I would also like to thank my promoters Sjaak Brinkkemper and John-Jules Meyer for their support and for the valuable feedback for the thesis. I am thankful to the members of my assessment committee Michael Luck, Anna Perini, Amal El Fallah-Seghrouchni, Munindar P. Singh, and Leon van der Torre, for evaluating this thesis and for providing valuable feedback.

I would like to thank Başak, Sercan, Jan, Brian, Natasha, Samarth, Parantapa for giving me the opportunity to conduct research with you. It was interesting and motivating working together and I hope we will have the chance to continue collaborating also in the future.

Thank you Lucas. You are a great friend and an amazing person. Thank you for sharing with me this whole adventure and for the invaluable time we had together. I will not write many words here because they wouldn't make justice to how grateful I am to have met you and to the value that I give to our friendship. Simply, thank you for coming into my life.

Thank you Jan. We had shared quite some intense moments in these years. Thank you for dealing with me and my frenetic mindset. Hopefully I didn't drive you too crazy! I found it stimulating to work with an intelligent and knowledgeable person like you. I'm sure you have a bright future ahead. Good Luck to you.

Thank you Başak. You have been an example for me during these years. I feel lucky to have met you. Thank you for your honesty and professionality, for the great advice you gave me many times, and for your friendship. I wish you all the best for your life and career.

Thank you Lorenzo for the brilliant cover of this thesis, it is an honour for me to have it designed by you, for I admire you and I'm always inspired by your creativity and talent.

I would like to thank all my colleagues in Utrecht. I apologize for not listing all your names, I would certainly forget to mention some. Thank you Remi, Onuralp, Can, Daphne, Ioannis, Federico, Jurian, Bilge, Vijanti, Bas, Silja, Marcela, Sergio. It was a pleasure to interact and share great moments with all of you. A special thanks goes to Samaneh. Thank you for all the enriching discussions and for sharing incredibly long days with me. PhD life can be sometimes very complicated and moody. I am grateful to have had a calm, solar and hardworking officemate like you.

A particular thanks goes to Kim, Colin, Oliver, Garm, Paola, Pradeep, Mohammad, Mostafa, Vahid, Duncan, for making every conference an unforgettable fully immersive experience.

I thank all the great people that I've met over the years in Utrecht.

Thank you Magdalena for always demanding quality of me, for continuously motivating me to do better, and for turning my life upside down.

Thank you Jacopo, Claudia, Julia, Matthieu, Camillo, Tania, Katerina, Christina, Laura, Serge, Francesca, and the others impossible to mention all here. Thank you for all the special days and nights together, for all the enriching discussions, and for inspiring and entertaining me. I have learnt a lot from all of you.

A big thanks goes also to the Lichtegaard crew. Bruna, Loreta, Fabio, thank you for helping me surviving the last year of PhD during the pandemic.

I would like to thank also all Italian friends. Federico, Lorenzo, Davide, Andrea, Davide, Matteo, Paolo, Fabio, Giorgia, Sara, Noemi, Enrico. Thank you for always supporting me, for tolerating my frequent disappearances and for accepting my madness. Having you in my life makes me feel home wherever I am.

Livio, please accept my heartful gratitude. Thank you for taking me under your wing, for everything you taught me, and for showing me what it means to passionately believe in something and pursue it.

Finally, an important thanks goes to my family, and in particular to my parents Sergio and Luigina, and to my brother Luca. Thank you for standing me and for always supporting my decisions with unconditional love.

Utrecht,
01/01/2021

Davide Dell'Anna

Curriculum Vitae

Education

2017 - 2021 **PhD in Computer Science**
Utrecht University, Dept. of Information and Computing Science
"Data-Driven Supervision of Autonomous Systems"
Promotors: M. Dastani, S. Brinkkemper, J.-J.Ch. Meyer
Co-promotor: F. Dalpiaz

2013 - 2016 **MSc. Degree in Computer Science**
University of Turin, Dept. of Computer Science
"Numerical and temporal planning for a multi-agent team acting in the real world"
Advisor: P. Torasso
Final mark: 110/110, honors and recommendation for publication

2010 - 2013 **BSc. Degree in Computer Science**
University of Turin, Dept. of Computer Science
"A mixed-initiative approach to resource contrained planning"
Advisor: P. Torasso
Final mark: 110/110 with honors

Academic Activities

2017 - 2020 **Teaching Assistant**
Utrecht University, Dept. of Information and Computing Science
Courses: Requirements Engineering, Methods in AI Research, Informatics Introduction Project, Intelligent Systems, Logic for Computer Science

2018 - 2020 **Students Supervision**
Utrecht University, Dept. of Information and Computing Science
MSc: Balancing Norms and Goals in Normative Agents (2020)
BSc: The Effect of Norm Enforcement on Traffic Congestion via SUMO Simulation with Norm Aware Agents (2018)

Involvement in Scientific Events

Organizing Committee

REFSQ 2018	Student Volunteers Chair of the 24th International Working Conference on Requirements Engineering: Foundation for Software Quality

Program Committee

RE4AI 2021	2nd International Workshop on Requirements Engineering for Artificial Intelligence
iMLSE 2020	2nd International Workshop on Machine Learning Systems Engineering

Reviewer for International Journals

Springer	Requirements Engineering
Springer	Autonomous Agents and Multi-Agent Systems
Springer	World Wide Web: Internet and Web Information Systems

Reviewer for International Conferences

REFSQ 2021	27th International Working Conference on Requirement Engineering: Foundation for Software Quality
PRIMA 2020	23rd International Conference on Principles and Practice of Multi-Agent Systems
RCIS 2020	14th International Conference on Research Challenges in Information Science
EMAS 2019	7th International Workshop on Engineering Multi-Agent Systems
AAMAS 2018	17th International Conference on Autonomous Agents and Multi-agent Systems
EUMAS 2017	15th European Conference on Multi-Agent Systems

SIKS Dissertation Series

2011
01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models
02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems
04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference
05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage
07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction
08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues
09 Tim de Jong (OU), Contextualised Mobile Media for Learning
10 Bart Bogaert (UvT), Cloud Content Contention
11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective
12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining
13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling
14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets
15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval
16 Maarten Schadd (UM), Selective Search in Games of Different Complexity
17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness
18 Mark Ponsen (UM), Strategic Decision-Making in complex games
19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles
20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification
30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization
39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games
40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control
42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
44 Boris Reuderink (UT), Robust Brain-Computer Interfaces
45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012 01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda
02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories
04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications
05 Marijn Plomp (UU), Maturing Interorganisational Information Systems
06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories
09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms
10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems
15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
28 Nancy Pascall (UvT), Engendering Technology Empowering Women
29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval
30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications

35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes
37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior
48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data
49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

---

2013 01 Viorel Milea (EUR), News Analytics for Financial Decision Support
02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics
04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
07 Giel van Lankveld (UvT), Quantifying Individual Player Differences
08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications
10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
12 Marian Razavian (VU), Knowledge-driven Migration to Services
13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications
16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid
18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams
37 Dirk Börner (OUN), Ambient Learning Displays
38 Eelco den Heijer (VU), Autonomous Evolutionary Art

39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games
41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning
43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

2014 01 Nicola Barile (UU), Studies in Learning Monotone Models from Data
02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
06 Damian Tamburri (VU), Supporting Networked Software Development
07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10 Ivan Salvador Razo Zapata (VU), Service Value Networks
11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
14 Yangyang Shi (TUD), Language Models With Meta-information
15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying
38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

2015
01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
03 Twan van Laarhoven (RUN), Machine learning for network data
04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments
05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning
31 Yakup Koç (TUD), On the robustness of Power Grids
32 Jerome Gard (UL), Corporate Venture Management in SMEs
33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
34 Victor de Graaf (UT), Gesocial Recommender Systems
35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016
01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments

16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UVA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees

13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TUE), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology

11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TUE), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TUE), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

---

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VU), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances

30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OUN), Learning visually grounded and multilingual representations

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TUE), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nobrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
30 Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
31 Gongjin Lan (VU), Learning better – From Baby to Better
32 Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
33 Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
34 Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development

2021 01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games for Social Interaction in Public Space
02 Rijk Mercuur (TUD), Simulating Human Routines:Integrating Social Practice Theory in Agent-Based Models