Utrecht University
Faculty of Science Artifical Intelligence

Approximating flexibility of social practices in a household setting

By Maaike Burghoorn

Supervisors: Dr. F.P.M. (Frank) Dignum[1], Professor Yoshihisa Kashima[2]

1. Utrecht University
2. University of Melbourne

March 15, 2019

Abstract

We aim to gain understanding in the factors that influence the daily pattern of social practices in a household setting. Studies suggest that household water and electricity use are generated by the interactions of the occupants within the infrastructure of the home. These patterns are often interlocked and difficult to understand. Learning which factors influence this interlocked pattern of practices can help understand how to shift everyday practices to be more sustainable. In this thesis, a simulation was developed that approximates the flexibility of these social practice patterns. The simulation uses agent-based modeling to simulate a negotiation protocol. Scheduling techniques were used to build the daily interlocked patterns. The simulation was used to examine the influence of schedule tightness and occupant flexibility on the interlocked patterns. It was found that the occupant flexibility seems to not influence negotiation at all and in general tighter schedules do not require more complex negotiation. Morning and midday tasks often did not require any negotiation, thus the tightness and flexibility did not have a significant influence in these events. However, households with a slightly busier schedule were more likely to require negotiation when modifying evening tasks. This is due to the local tightness during the evenings. The results suggest that shifting towards more sustainable living is not constrained by schedule tightness or the willingness of people to change.

Acknowledgments

I would like to extend my thanks to Dr. F.P.M. Dignum for giving me the opportunity to do my master's thesis in Australia. I would also like to thank him for his advice and assistance in keeping my progress on schedule and his useful criticism on my thesis.

I would like to express my great appreciation to Professor Yoshihisa Kashima for his valuable suggestions during the development of this thesis. He always found time for our meetings in his busy schedule. I always looked forward to sharing my progress and ideas and listening to Yoshi's thoughts and view.

I would also like to thank everyone in the Social Psychology department of the University of Melbourne who joined the weekly social psychology group sessions. It was great listening to all your researches and accompanying hardships. Thank you for listening to mine as well.

My special thanks are extended to Curtin University Sustainability Policy institute (CUSP). Thank you for sharing your time, knowledge and for providing me with useful data for this thesis. Special thanks to Professor Greg Morrison and his family for their efforts and providing me accommodation.

Contents

1 Introduction

In Western Australia water resource use is a big issue due to the Mediterranean climate which causes a water shortage. Research has been pursued to gain understanding in the behavior and technology that influence sustainable living. It was suggested that water and electricity use is not solely dependent on the infrastructure of the home since the individuals living in the home are responsible for the emission of the home (Eon et al., 2018c). Technologies have been invented to provide occupants useful information regarding their resource use. However, these technologies often fail to become integrated in the daily life of occupants and lose efficiency after a while. It was also proposed to solely focus on influencing the behavior and intentions of the occupants. Lopes et al. (2012) has demonstrated that this approach can achieve significant energy savings. However, generally individuals are not willing to change their personal behavior because this requires everyday effort which diminishes their living comfort.

Social practice theory shifts the behavior change question from "How do we change individuals' behavior to be more sustainable?" to "How do we shift everyday practices to be more sustainable?" (Spurling et al., 2013). This shift in approach is what recent studies have been researching. Studies at the Curtin University Sustainability Policy Institute (CUSP) are aiming to combine human behavior with the infrastructure of the home. It was proposed to view the home as a system of practices to gain a deeper understanding of the complexities of the home system (Eon et al., 2018a). This approach uses the idea that emissions are generated by the interactions of the occupants with the infrastructure of the home. These interactions are the daily social practices of all occupants in the household. All these practices affect the total resource use of the household. In Spurling et al. (2013) it was argued that the pattern in which these practices are performed is interlocked. Slightly adjusting one practice sometimes requires many changes in this pattern. Macrorie et al. (2015) argued that to understand changes in practices, it is required to understand not only the singular practice entities but the connections among practices across both space and time as well. Practices interlocking can occur through sequences of practices and synchronization of practices. The former forces an order in which practices are performed, such as working day hours or shop opening hours. The latter causes interlocking by people performing the same activities during the same period of time, such as driving to work or showering before work.

Schedules of social practices within households differ in flexibility. This depends on a number of factors, such as number of people within a household and the activities of each occupant. However, it also depends on the specific meanings behind those activities. Besides the meaning, this pattern was also shaped by the available resources and skills of the occupant. Social practices stem from a theory within the sociological research field. Social-psychologists think of social practices as the primary focus and the involved individuals are mere carriers of these practices. However, within agent-based modeling and more specifically social simulation, social practices are seen from the individual's perspective. The practices still have a meaning attached and require skill and resources. These meanings, skills and resources serve as constraints for duration and timeslot in which a practice can be performed by the individual. Since agents have to perform many social practices on daily basis, the pattern of these practices often becomes interlocked and complex. Therefore when an agent has multiple meanings, skills and resources attached to its social practices, this pattern becomes more difficult to change. Moving just one practice can cause a chain reaction of changes within the schedule. This is due to the complex connections among all practices. Practices can share meanings, skills and resources with other practices making them interdependent.

In this thesis, we are testing the flexibility of these patterns in households. For a human eye, it is often easy to see which changes are necessary to accommodate one small change in the schedule. Humans are intuitively able to view the meaning, required skills and resources behind the practices. However, for a machine handling these constraints is a difficult task.

The meanings, skills and resources need to be built in as constraints for the machine to know which moves are legal. In this thesis, we aim to build a social simulation that models the social practice patterns within households. Using this simulation, we try to approximate the flexibility of interlocked patterns of social practices in households. The simulation is based on agent-based modeling techniques. We also use a technique called scheduling, which simulates the interlocked patterns of practices within households. The scheduling technique is used to create an initial schedule of practices that fulfills conditions. These conditions can be preferences of the occupants, such as their preferred shower time. Other conditions consist of meanings, skills and technology which are required to successfully perform a practice. The resulting schedule is used as the default daily schedule. This basic schedule compares to System 1 behavior or fast thinking as it allows people to follow some kind of pattern to avoid actively thinking about ones actions each day, (Kahneman and Egan, 2011). The agent in the household will be given the opportunity to modify their schedule. This process is handled by the use of a negotiation protocol.

This thesis project is incorporated within the low carbon living project at the University of Melbourne. This project studies households in Australia and the likelihood of changing their behavior in favor of low-carbon living. This research draws some of their ideas from social practice theory and how meaning, skills and resources can influence people's decisions and behavior towards low-carbon living. In this thesis we want to zoom in on a small subsection of sustainable living, which is the water use and more specifically the showering practice. The shower practice is shown to be the second biggest contributor of water use within the household after garden irrigation, (Eon et al., 2018a). In this paper it was also shown that there exists a lognormal distribution in the frequencies of the lengths of the showers. This distribution can be interpreted to represent the different meanings of showers with different durations. The different meanings make the showering practice an interesting application for the practice scheduling model. Multiple meanings imply the different types of constraints which can interlock a daily schedule in different ways. The shower practice can be used as a representative example of a social practice with many constraints attached. Therefore, this thesis will focus on shower practices in particular.

Research questions

Based on this problem description and the involvement of the low carbon living project, the developed simulation will be used to answer the following research questions:

- How do schedule tightness and occupant flexibility influence the interlocked pattern of social practices in a household setting?
- How do schedule tightness and occupant flexibility constrain the shower practice?

In Section 2, we present related work from both the Social Sciences field as well as the Computer Science field. We will discuss studies towards agent-based modeling, social practice theory and scheduling theory. We give an extensive description of the model developed for the purpose of this thesis in Section 3. This section mainly describes the agent-based negotiation protocol. Details on the scheduling protocol are explained in Section 4, which will be preceded by a brief introduction to scheduling theory. The setup of the experiment which is used to answer our research questions, is discussed in Section 5. This followed by the results of the experiment in Section 6. Finally, we will discuss the implications of the results in Section 7 and use these implications answer the research questions, draw some conclusions and indicate possible extensions of the model in Section 8.

2 Related work

Many studies have attempted to implement a social aspect in agent-based models (Athanasiadis and Mitkas, 2005; Luo et al., 2008; Marsella et al., 2004; Mercuur, 2015). It has become more important for agent-based systems to be socially aware. An agent must be able to understand his social role within the social context and choose appropriate actions in given situations. Most attempts that add a social context to their agent-based model use a fixed context. If the social context were to change, the agents within the system are unable to reason about their social roles in the new social context. Therefore, a context change often requires a complete re-engineering of the system.

Several studies attempted to generalize the social deliberation system of the agents (Dignum et al., 2015; Dignum and Dignum, 2014). Such a system allows agents to deduce their social role given a social context and decide on appropriate actions. The difficulties lie in finding a way to represent this social context and build a model which lets agents deduce social norms, values and other social concepts from the context. Several papers have tried to represent the social context using social practice theory and have created a social deliberation model using this theory.

Social practices stem from a sociological theory called social practice theory (Shove et al., 2012). The theory seeks to determine the link between practice and context within social situations. Social practices are defined as activities that are performed by individuals on a regular basis. These activities are a shared concept meaning, they are common activities among individuals in society. However, each social practice can be performed differently as they are dependent on the individual who performs them. An example of a social practice is "going to work". Many individuals perform this practice on daily basis, but depending on the individual one might drive to work or use the public transport.

Dignum and Dignum (2014) argue that social practice theory forms an efficient way to represent social context because it connects practices to context. Social practices integrate the individual with its surrounding environment, which allows the individual to assess how the current social situation relates to its past experiences, capabilities and culture. In Dignum and Dignum (2014) the social practice theory was used as the focal point of the deliberation process of the agents. A model was developed which simulates the action planning process of an agent using social practice theory. In Dignum et al. (2015) this model was extended to interaction situations and several social aspects were integrated with social practices, such as norms, self identity, and habits. The social aspects allows the agent behavior to be consistent overtime.

Social practice theory is also widely explored in other areas. Recent studies in the sustainability field are using social practices to study resource use of households. Especially in Australia a lot of research has been conducted in this area. The University of Melbourne is currently working on a low-carbon living project (O'Brien et al., 2017). This project studies households in Australia and the likelihood that they will change behavior in favor of low-carbon living. This research draws some of their ideas from social practice theory and how meaning, skills and resources can influence people's decisions and behavior towards low-carbon living.

2.1 Agent-based modeling

Agent-based models (ABM) are computational models that simulate the actions and interactions among autonomous agents in an environment. ABM allows the user to model the emergence of macro behavior by simply modeling individual decision and interaction behavior. The possibility of modeling these interactions is the main difference that separates agent-based modeling from other computational models (Gilbert, 2008). Agent-based models are often used to model complex phenomena and study the influence of numerous factors. In Schelling (2006) it is explained how minor changes in the micro motives can cause a large effect in the macro behavior. Minor motives is defined as the behavior on the individual level and the macro behavior is the

resulting aggregate behavior. An example mentioned in Schelling (2006) shows that individuals with a slight racial preference can result in completely racially segregated neighborhoods.

An advantage of ABM is that an experiment can be repeated many times, while varying several parameters. The ease of switching parameters allows for testing under different conditions, while all other factors are kept constant. The simulation is isolated and all observed effects can be explained by the modeled parameters.

Agent-based models consists of agents that interact with their environment. Agents are individual entities that often follow simple decision rules. These decision rules often consist of a policy function with the aim to optimize the agent's utility. Therefore, the agents are often rational entities, which are solely self-interested. However, in more social systems agents have to collaborate to achieve a shared objective. These models require communication among agents, which allows agents to share information with other agents. Most agent-based models incorporate learning. In these models, agents learn from past experiences and update their decision rules accordingly. Much research has been done towards developing learning algorithms such as Reinforcement learning (Sutton and Barto, 1998), Bayesian learning (Young, 2004) and Satisficing strategies (Stimpson et al., 2001).

The virtual world in which the agents are contained is called the environment. This environment can be spatial, where the agents have coordinates to indicate their location. In a spatial environment the interactions among agents are usually limited to a specific range. In some models agents are not allowed occupy the same location as other agents. In other models agents might positively or negatively influence specific behavior of their neighbors. Another type of environment is networks. Agents within networks are connected by network links (Gilbert, 2008). These networks can represent different types of relationships among agents, such as family or customer relationships.

Agent-based social simulations is a branch within agent-based modeling which concerns itself with modeling social behavior. In these simulations, the agents represent persons or a group of persons. Modeling human behavior is quite complex because social interactions depend on many different factors. It is therefore impossible to create a social simulation which completely and accurately models the reality. However, the aim of social simulation is to create a simplified representation of "social reality" that encodes the way in which reality is believed to operate (Gilbert, 2008). A set of simple rules can often closely represent the behavior in social reality.

In social simulation it is important that the agent or entities are able to communicate with each other. One type of communication is negotiation, where the agents try to reach an agreement. It is common for one agent to initiate the negotiation by sending a request to the other agents. This thesis will use negotiation among agents to handle modification of schedules. The negotiation protocol used in the model was inspired by Mester et al. (2015). In this study a negotiation protocol, PRINEGO, was developed that allows agents to argue about their privacy expectations. The protocol allows users to negotiate about their preferences regarding post sharing on social media. It handles the flow of messages between negotiating parties. Agents can either accept or reject a post request and negotiation continues until the majority of the privacy expectations is met. Mester et al. (2015) does not use the concept of counter-offers, as it is difficult to compare offers based on their privacy. However, in utility based negotiation, counter-offers are commonly used.

2.2 Social practices

Society is full of habitual practices that are part of daily life. A few examples of these practices are "showering", "cleaning" and "going to work". These practices can be performed in many different ways depending on the social context of the practice but also on the individual who performs the practice. We should mention that social practice theory is a theory within sociology. The practice itself is seen as the focal point, whereas the individuals are mere carriers of these practices (Shove, 2010). The theory seeks to determine the link between practice and

context with social situations. Therefore, practices cannot be seen as deterministic entities but should be viewed as shared (social) entities (Shove et al., 2012). People share practices in the sense that everyone is familiar with the concept of "cleaning". But how, when and where this practice is executed depends on the individual. One could see a social practice as an elaborate condition-action rule (Dignum et al., 2015). A social practice is triggered by the context. One might find a messy house that requires cleaning, another might assess the situation as a comfortable mess or not messy at all. Once a social practice has been identified, it is concretized by filling in details. Dependent on which part of the house needs to be cleaned, a vacuum cleaner or a mop might be required. More generally speaking, filling out the practice results in specific actions dependent on multiple factors. Researchers in the social science (Holtz, 2014; Reckwitz, 2002) have categorized these factors into three broad categories: meaning, skill and technology

1. Meaning: The reason attached to the social practice. It specifies why a practice is triggered and what objective the individual that carries the practice is trying to achieve.

2. Skill: The required competences that are necessary to perform the practice. An individual might require a certain level of intelligence or a practice can require multiple individuals at the same time.

3. Technology: The resources that are required to perform the practice. These resources can be finite and/or shared with other practices or individuals.

The way these three elements of social practice are filled in can depend on social context, habits, norms and social identity (Dignum et al., 2015). Habits refer to practices that are performed frequently and do not require active thinking from the involved individuals. This corresponds to System 1 or fast thinking. Actions that require a more deliberate process before they can be executed, correspond to System 2 or slow thinking (Kahneman and Egan, 2011). Social identity refers to how individuals perceive themselves and the social world (Dignum et al., 2015). This allows for different meanings to be attached to a (shared) social practice. Once these different meanings, required skills, and technologies are specified, the practice can be carried out. Social practices can evolve over time and new meanings, skills and technologies can be added to the practice.

2.3 Social practice theory in agent-based models

In this subsection we discuss some of the studies that have applied social practice theory to agent-based modeling. In all of these studies, social practice theory is used as the foundation of the deliberation process. However, different concepts are used to fill in the social practices and select the suitable actions. The different studies will also show that social practice theory in agent-based models can be applied to a diversity of social situations.

In many recent applications, agent-based models need to be aware of the social context. This context is required for an individual to plan its goal and actions accordingly. A social context concerns all surroundings of the agent, including other agents. Agents which can act accordingly are agents that are capable of adapting their plans and actions given the social context. However, in many social models the contexts are usually fixed. If a new context was fed to an existing model, the agents in this model would not be able to reason about their social roles in the new context. The reason is that agents are often implemented to operate in one specific context. Therefore they are incapable of understanding new contexts without major changes to their deliberation system. Several studies have attempted to solve this problem by developing a generic deliberation model that interprets social contexts using social practice theory.

In Dignum and Dignum (2014) it was proposed to use the social practices as heuristics for planning in social contexts. It is explained that several previous studies have added social

practice theory to their deliberation model, but always as an additional aspect. Dignum and Dignum (2014) propose to use social practice theory as the foundation of the deliberation rather than an extra filter at the end of the deliberation process. The paper describes two types of behavior, reactive behavior and pro-active behavior. The former concerns behavior in habitual circumstances and is similar to System 1 thinking. The later requires socially intelligent behavior to analyze the social context. Most agent platforms only use one of the two types of behavior, however in a social model both types of behavior are required.

Dignum and Dignum (2014) argue that social practice theory can be used to combine both types of behavior. Social practice theory concerns habitual practices which are performed on daily basis. They can be seen as blueprints which can be filled in by many different actions depending on the context. Every time a practice is used, previous encounters of the practice, the current environment, meanings and purposes of practice are adapted. Hence, both reactive and pro-active behavior is used in the deliberation process. The paper also argues that since social practices are shared concepts, they are very well suited to represent individual planning in social simulations.

The final model in Dignum and Dignum (2014) defines several components of a social practice that can be concretized. We will not describe each characteristic in detail, but rather explain how these characteristics help to fill in the social practice. Fast thinking is used when a social practice matches the characteristics of a situation and a habitual action can be triggered. If the situation does not match previously used social practices, more deliberation is required. The social practice is filled in by sensed observations, which concretizes the social practice until an appropriate action can be deduced. The paper illustrates their model using an example in which fire brigade agents have to manage a disaster situation. The situation involves a collision between a truck and a passenger car which consequently caught fire. Analyzing the context results in filling in the social practice "tanker fire". Depending on how the social practice is filled in the action "evacuation" or "extinguish fire" might be triggered and performed. This example scenario shows one of the many applications in which social deliberation agents can be useful.

Dignum et al. (2015) is another paper that uses social practices as the foundation of the deliberation process. This paper focuses on a scenario in which interaction among agents is required. It uses several concepts such as social identity, norms and values to create consistency in the resulting actions of the deliberation process. The concept of social identity is described as the capability to perceive itself in the social world. The identity is represented with a set of values which ensures a consistent type of behavior. The concept of norms is used to denote which actions are (dis)allowed in certain contexts. This again provides consistency in actions over time. Habits are actions that are performed frequently and are another way to create consistency in the model. The paper describes how social practices can be filled in using these concepts. Social identity is used to give a social meaning to a practice. The available actions are filtered based on norms and habits are formed when a social practice is repeated frequently.

Dignum et al. (2015) presents a business case scenario. In the scenario a large company and a smaller research institute negotiate business terms. Both parties portray completely different self identities, which shows the necessity of using this concept in the deliberation process. A rough deliberation cycle that uses self identity and social norms is presented. It shows how these two concepts can be used and how habits can eventually be formed. First a selection of self identity values is made based on the current situation. This self identity is used to identify all applicable abstract social practices and the most suitable one is selected. Hereafter, the social practice is filled in and an action is selected based on the concrete social practice. Finally the action is executed and the results of the actions are evaluated. Social practices are adapted based on this learning process. The self identity and norms are used both in selection process and during instantiation of the social practices.

Another scenario example of applying social practice theory to an agent-based model is

presentend in Mercuur (2015). The illustrated model depicts meat-eating behavior while dining out. It demonstrates the deliberation process of eating meat or eating vegetarian. It is argued that social practices create an well-balanced combination of habitual and intentional behavior, unlike traditional theories which adopt only either of the two. All three main elements of social practices introduced by Shove et al. (2012) are used in the model: meanings, skills and technology. Meanings of dining out can include pleasure, health or necessity. Required skills might be etiquette and usage of cutlery, while cutlery itself and other tableware can be categorized as the required material.

2.4 Social practice theory in households

In the 2.2 subsection we already explained the general concept of social practices. Practices can range from "going to work" to "cleaning". But in this research we are particularly interested in practices that are performed in and around the house. This includes the "sleeping" and "showering" practice but also practices such "working". The latter practice is not a household practice itself, but it does influence the water and energy use in the household as shown in multiple studies (Eon et al., 2018a,c). Working occupants leave the household during the day and therefore the energy and water use is minimal during these hours. Recently, several studies have focused on social practices within the household. Many of these studies are conducted in Western-Australia due to the Mediterranean climate, which causes a water shortage. We will discuss some of the relevant studies here.

Before the focus shifted to social practices, the household was viewed as a single entity (Eon et al., 2018c). In these models, energy and water use is only dependent on the infrastructure and technology of the household. Technology has been incorporated in households to make occupants more aware of their energy use. However, if this technology does not become embedded in occupant everyday lifestyle, the technology is forgotten after a short while. However this way of modeling does not match the energy use in actual houses. In Eon et al. (2018c) the daily heating and cooling practices of occupants were studied. It was shown that households with similar designs can still show very different patterns of energy use. This was due to difference in choice of appliances and technology in the households. Another influencing factor was the difference in lifestyle. This includes structure of the family in the household and their daily practices. The study also found that lifestyles within households differed as much as lifestyles between households. Different occupants may have different beliefs and therefore take different actions regarding the same practice. This supports our idea to constraint tasks of occupants based on the meanings attached to the practice.

In a follow-up research (Eon et al., 2018b), the influence of occupants and their practices on resource use is also studied. In this study the homes are viewed as a system of practice (SOP) to understand the resource and technology use in the home. This supports the idea that household emissions are not dependent on the physical infrastructure of the house alone. Instead they focus on the interaction of the occupants with the home and how practices are filled in differently by different occupants. Occupants can attach different meanings, places and contexts to these practices, which causes different interlocking patterns of practices. The term interlocking is used to indicate the mutual dependency between daily practices. This means that practices are constraint by their contexts and other practices. Results showed that changes in practices were considered too much work by the occupants. However, a change in technology was seen as a simple one-time solution. Occupants reported that changes in practices causes too much discomfort, such as reducing the shower time. The showering practice sometimes serves as a relaxation purpose, which makes it difficult to change this task. This again shows that meanings constrain the practices and therefore our model will use meanings as constraints.

Another paper by Eon et al. elaborates on the SOP approach (Eon et al., 2018a). The study applied the SOP approach to eight Australian homes, which incorporates human interaction with other occupants but also with the technology in the household. This approach gains

a deeper understanding of the interlocking patterns within households, which is necessary to further reduce resource use. The study found that practices are influenced by interlocked practices and interlocking routines of other occupants. It was also shown that practices follow established daily patterns. These results imply that highly interlocked practices are not easily susceptible to change. However automation might slightly dis-interlock these practices from the established daily pattern. Furthermore, the study shows that daily patterns can be re-aligned, for example weekdays exhibit different social practice patterns than the weekend.

2.5 Sustainability and application

This research project is part of the sustainable living project at the University of Melbourne. For that reason a sustainability setting was chosen to test our model. The University of Melbourne is developing a social-psychological decision-making framework. They study the factors that influence the transition to low carbon living. The paper argues that an individual's daily activities form a network which balances one's own resources but also the activities of others. Moving one activity can disrupt the balance of the entire network, which makes behavior change a difficult task. A measure, called the Low Carbon Readiness Index (LCRI), was developed which is used to identify the likelihood of transitioning to low carbon living. Different types of mindsets towards low carbon living were identified and behavior was clustered to categorize different types of change that accommodate more sustainable living. This allows the LCRI measure to predict what sort of sustainable behavior change each type of mindset is likely to show.

3 The Model

This section provides an extensive description of the model developed during this thesis. The details of the model will be discussed such as the overall process overview, design concepts and used datasets. Each entity used in the model will be explained along with their initial instantiation and update rules. This section provides an insight in the working of the model, design choices of the model and allows for replication of the model. The model description follows the ODD (Overview, Design concepts, Details) protocol (Grimm et al., 2006, 2010).

3.1 Purpose

The purpose of this social simulation is to simulate negotiation among occupants living in the same household concerning their daily activity schedule. The simulation runs the negotiation process of multiple different type of households at once. However, the households do not influence the negotiation process of other households. Each household is equipped with a basic daily schedule that covers the daily activities of each member in the household. This schedule represents the habitual pattern in which people tend to live. However, this habitual pattern is not fixed and allows changes when required. The schema of each occupant is dependent on the schemas of the other occupants. Therefore changes to one person's schedule requires negotiation with the other household members. This social simulation models this negotiation process of alternation of the basic daily schedule. The influence of several factors is measured, such as the tightness of the schedule.

3.2 Entities, state variables, and scales

This model consists of two parts: the scheduling of tasks within a household by a global mechanism and the negotiation of changes to this schedule. The later part concerns the agent-based model with agent-like entities. However, we will explain the entities that appear in both the parts of the simulation as the entities used in the agents part were designed to allow scheduling by a global mechanism.

Persons

The Person entities represent the agents of the model. They are the occupants that live in households and they are the entities for which a schedule is created. In the social simulation component, these are the entities that negotiate with each other to modify to the schedule. Each Person has a unique ID within their household and a Household object which defines the household in which they live. A Person owns a list of tasks which represents their daily activities. These tasks are the daily social practices of the agent. The list is filled with Task entities. Besides this list of Tasks, a Person also owns a list of TaskAssignments, the difference between the two entities will be explained below. It should be noted that the TaskAssignment list describes the daily schedule of a Person, while the Task list simply defines the daily activities of a Person. The difference being the time attribute.

A Person can be of three different types: Adult, Student or Child. These three subclasses differ in a number of attributes. For example, a Person's bedtime is set to always be scheduled between 21h-24h. However, if the Person concerns a Child his/her bedtime is constrained to the interval 20h-22h. By default, the risetime of a Person is set in the range of 6h-11h. However, if a Person has a full-time job, he/she is required to get out of bed before 8h. Similarly, when a Person has a Study task in his task list, he/she is required to get up before 9h.

Another varying attribute is the job type. This categorical variable can be set to full-time, part-time or unemployed. The later category covering both stay-at-home parents, housewives/husbands and retirees. Full-time jobs can only be assigned to Adults. An Adult can

also work part-time or be unemployed. A Student or Child can also have a part-time job or be unemployed, but they are never assigned a full-time job. Unlike the name suggests, Students are not the only subclass with a Study Task in their daily schedule. Some Children attend school and therefore also have a Study Task. An Adult never has a Study Task as adults that attend schools are categorized as Students.

The remaining variables are independent of subclass. The preferred shower time is the only variable that is set at random within the Person class. This categorical variable specifies if a Person prefers to shower in the morning or evening or has no preference at all. The morning preference ranges from 6h to 11h, while the evening preference covers the range 20h-24h. The final three attributes of the Person entity are a RuleBase object, a flexibility parameter and a flexibility increase. The RuleBase object can be seen as a database in which the negotiation rules of a Person are stored. A Person entity uses the rule-base during negotiation to respond to negotiation offers. This object is identical for every agent, as everyone obeys the same rules. The flexibility parameter specifies how flexible a Person is in making changes to his/her daily schedule. When the Person is requesting change he/she is more flexible than when the Person is responding to change. This increase in flexibility is set by the flexibility increase parameter.

TimeSlots

TimeSlot entities are used to represent time within schedules. A TimeSlot object has a unique ID to distinguish it from other TimeSlot objects and a starting minute attribute. The starting minute attribute represents the number of minutes that have passed since midnight. For example, 06:00h is represented with a starting minute value of 360 and a value of 1020 indicates the time 17:00h.

One day is represented by 96 TimeSlots, one object for every 15 minutes of the day. They represent the available starting times for the daily tasks. The solver schedules a Task to start at the starting time of the TimeSlot it is assigned to. TimeSlot objects also have a Person attribute. This attribute is not instantiated upon creation, because a TimeSlot object can be assigned to multiple TaskAssignments during the scheduling phase. After the scheduling phase, these 96 TimeSlot objects are replaced by $n * 96$ objects, where n is the number of people in the household. Each TaskAssignment is assigned a unique starting TimeSlot object, which also allows the TimeSlot person attribute to be instantiated. The reasoning is explained in Section 3.4.

Households

A Household object is a collective of Persons. It represents the living environment of these Person objects which are otherwise referred to as occupants. The Household has a unique ID and stores the Person objects in a list. Besides the occupant list, a Household contains a list of TimeSlots, Tasks and TaskAssignments. The TimeSlot list stores all the available TimeSlots for each Person. The Task list stores all the tasks of each Person and for each Task a TaskAssignment is stored in the TaskAssignment list. The latter can be seen as the unassigned schedule of the household. After the solver assigned a starting time to each TaskAssignment in the list, the schedule can be visualized using the starting times and durations of Tasks.

A Household can have 1 to 5 occupants, this depends on the hType attribute. This attribute specifies the type of the household. There are a total of 24 different households, which can be read from the data (see Section 3.6).

Tasks and TaskAssignments

A Task object describes a daily activity that a Person can plan during his/her day. A Task has a unique ID within the Household and belongs to only one Person. This Person object

is attached to the Task as an attribute. Every Task has a release- and duedate. These two dates respectively depict the earliest time the task can start and the latest time the task can be completed during the day. A Task also has a processing time which is the duration of that task in minutes.

A Task can be of eight different types: ShowerTask, WorkTask, StudyTask, SportsTask, PartyTask, DinnerTask, BedTimeTask and RiseTask. Each type represents a different type of daily activity and specifies its own limits of releasedates, duedates and processing times. They are also constrained differently by the solver, as specified in the rules. These rules are explained in Section 4. During negotiation the different task types are also constrained differently. Task objects have a shrinkable and pushable trait, which are both boolean variables. When a task is shrinkable its duration can be reduced during negotiation. Similarly, pushable allows relocation of the task within the schedule. The instantiations of these variables for each task type are indicated in Section 3.5.

ShowerTasks have a meaning attribute which specify the reason for the task. The meaning can be set to one of the three specified types: freshness, cleanliness and relaxation. Initially, these meanings are generated randomly but during negotiation a specific meaning is attached. This meaning directly influences the duration of the showers. A correlation between the meaning and the duration of showers is assumed, as implied by Eon et al. (2018a).

The starting time of a Task is not specified within the Task entity itself. Instead, TaskAssignment objects were created to specify this. The TaskAssignment entities contain a Task object and a TimeSlot object. This TaskAssignment entity functions as a connection between a Task object and a TimeSlot object. Therefore a TaskAssignment object assigns a task to a starting time. This design choice will be explained in Section 3.4. A TaskAssignment object has a unique ID within the household and matches the ID of the corresponding task. The object also contains a Person object because a TaskAssignment can only belong to one individual.

Events

The Event objects are generated after the scheduling phase and therefore are not part of the schedule. Instead, they are used to make changes to the basic schedule during the negotiation phase. An Event describes adjustments to one or multiple TaskAssignments in the basic schedule. These alterations represent sudden developments that can occur in real-life households, such as an earlier meeting at work or an extra sports activity. We have chosen to only generate events that require changes to the shower task, as this task is our main interest and serves the purpose of our research. However, some adjustments to other tasks are also required such as an extension of the working time or adding an additional task to the schedule.

Each Event belongs to one Person, whom we will refer to as the requesting agent. This Person was randomly selected from the Household and added as an attribute to the Event. Similar to the other entities, Event objects also have a unique ID. Besides the Event ID, the ID of the changing TaskAssignment is also stored. This TaskAssignment always concerns a ShowerTask that has to be rescheduled. In some Events a subtask object is included. This subtask is an extra task that is requested to be added to the basic schedule, e.g. an Sports Task. If no extra Task object is to be added, the subtask attribute is null.

The requesting agent proposes a starting time and duration of the new ShowerTask, these details are stored in the Event. Another preset attribute is the maximum time to subtask. This variable only allows scheduling of the ShowerTask within a specific time range of the subtask. The final preset attribute is a boolean that indicates whether the ShowerTask should be scheduled before or after the subtask. For example, the shower should occur after the Sports Task but in the case of a Party task the shower should be planned before the subtask. The time at which this subtask starts or ends, depending on this before or after constraint, is stored in a time constraint variable. This time constraint is the time before or after which the ShowerTask should start.

A few attributes were added to the Event object to use during the negotiation process. The reduction attributes stores how much the duration of the requesting agent's ShowerTask should be reduced to fit in the schedule. During negotiation, a list keeps track of the TaskAssignments that were adjusted to fit the Event. This list consists of CounterOffers, which will be explained below, and is used to update the schedule with all the required modifications when negotiation is successful. A final attribute keeps track of the number of times scheduling of the Event was tried at different time locations.

An Event can be of three types: earlyMeeting, eveningParty or workOut. The EarlyMeeting event requires an increase in the duration of the WorkTask. The starting time of the WorkTask is advanced by an hour and the end time remains constant. The ShowerTask should be scheduled before the WorkTask. The eveningParty schedules an extra task during the evening of the negotiating agent. This extra task is a PartyTask. Similar to the earlyMeeting event, the ShowerTask should be scheduled before the PartyTask. The final event type, workOut, requires the ShowerTask to be scheduled after the subtask. This subtask concerns a SportsTask which is always scheduled in the evening.

Responses

During negotiation the other occupants in a Household respond to the requesting agent's offers. They answer these offers by sending a Response object back to the negotiator. This Response object stores the Person object of the responding agent, as well as an response code to the offer. The code is binary: "Y" for accept and "N" for reject. If an offer is rejected, the responding agent also sends a counteroffer. This counteroffer is made by adding a CounterOffer object to the Response.

CounterOffers

The CounterOffer object used in Responses contain the ID of the TaskAssignment the requesting agent is willing to change. This TaskAssignment is always owned by the responding agent and a CounterOffer can never propose to change another occupant's activities. Besides the ID, a starting time and an ending time are also stored in the CounterOffer. These times suggest to move the responding agent's conflicting task which potentially resolve the overlap. The new starting- and ending time could imply a relocation of the task, a decrease in duration of the task or both. However, if the agent is inflexible the reduction might be zero and the agent will return its current times.

3.3 Process overview and scheduling

In algorithm 1 the main method of the simulation is shown in pseudo-code. In this function the creation of a schedule by the solver and the negotiation of modifications to the schedule by the agents are handled. The first step in this method builds a solver that can be used to schedule a list of TaskAssignments using constraint satisfaction. More on this process is explained in Section 4. Once the solver has been built, the shower duration distribution is read from a datafile. This data will be used during creation of Household objects to assign sensible shower durations to ShowerTasks. The number of Households to be created, solved and negotiated is read from a slider set by the user in the experimentation environment. A for-loop is initiated to handle the Households one by one. Consequently, the Household objects are handled in a fixed order. However, this is irrelevant to the process because Households do not interact or influence each other. At the end of this loop, the scores and the schedules created during the process are written to output files.

A Household object is created using the generateHousehold method, which will be elaborated in Subsection 3.7. Subsequently, the Household is handed to the solver, which determines and

returns a schedule for the Household. The score of this solution is returned by the calculateScore method. This method also provides the names of the violated hard- and soft-constraints. If the returned schedule is optimal, the list of violated constraints is empty. Hereafter, the tightness of the score is calculated. Once the schedule is fully processed, each Person in the household is assigned his/her Personal schedule, which only includes their personal TaskAssignments.

The next step initiates the social simulation phase. Unless the event is specified by the user, a random event is selected. The eligible agent is selected to negotiate the event. Eligible refers to e.g. full-time working agents when an earlyMeeting event is requested. The selected person will negotiate necessary modifications to the basic schedule. This negotiation process is managed by the handleEvent method. The required adjustments are stored in the event object and applied to the basic schedule in the applyChanges method.

This simulation does not use a time unit such as steps or ticks. Instead it uses TimeSlot objects to represent time units within the schedules. As presented on page 14, these objects represents the available time of an agent and can store the starting time of a task.

Algorithm 1 Main algorithm

```
1:  function MAIN
2:      solver ← buildSolver()
3:      READ shower data from file
4:      nrHouseholds ← nrHouseholds slider
5:      for nrHouseholds do
6:          household ← generateHousehold()
7:          solvedHousehold ← solver(household)
8:          score ← calculateScore(household)
9:          tightness ← calculateTightness(household)
10:         for p in Persons in household do
11:             p.schedule ← household.getPersonalSchedule()
12:         event ← generateEvent(solvedHousehold, eventType)
13:         SELECT random eligible person
14:         event ← person.handleEvent(event)
15:         negotiatedHousehold ← applyChanges(solvedHousehold, event)
16:     WRITE scores to file
17:     WRITE household schedules to file
```

3.4 Design concepts

3.4.1 Basic principles

The model is based on two underlying theories. The first theory that is used is the scheduling theory, which is used to build the daily schedules for the agents. The theory uses the concept of constraint satisfaction theory which tries to solve a resource allocation problem while adhering to several hard and soft constraint. Further details on this scheduling theory can be found in Section 4. The second theory is agent-based modeling, which is used to simulate the negotiation protocol among agents. The theory describes agents sending requests and responses to other agents. Mester et al. (2015) designed a refined agent negotiation protocol (PRINEGO) on which the negotiation protocol used in this thesis is based. Both theories are used as two major components of the simulation. They form the basis of the model.

3.4.2 Emergence

The layout and tightness of a schedules is hard to predict. The number of persons within a household, as well as their occupation, is preset in the data. Therefore, the daily tasks assigned

to each person are also fixed. However the duration of these tasks is random which introduces some unpredictability. Similarly, release- and duedates are also preset for each task type, but these intervals are still large enough to allow variation. Some task order constraints were preset as well, but these are very basic constraints and concern mostly the Bedtime- and RiseTasks. Despite these preset variables, variation exists to the extent that emergence can occur. Each set of tasks can create different solutions each time.

The scheduling process is not controlled by the agents themselves but only constrained by their properties. On the contrary, during negotiation agents' behavior does model the output. The agents communicate to modify the basic schedule, which can vary in complex ways depending on the characteristics of the agents and the basic schedule. The negotiating agent and the type of event are selected at random. Once an event type has been determined, an eligible agent is selected. For example, when a EarlyMeeting event is generated only full-time workers can be selected as the negotiating agent. The time at which the event is scheduled depends on the current basic schedule. The early meeting event advances and extends the current working time by exactly one hour. The shower is then scheduled before this event. Whether the latter operation is successful depends on when the current working task is scheduled and on the schedules of the other agents. If overlap occurs during negotiation, the flexibility trait of the other agents is an important aspect that influences the output schedule. If an event could not be fit at the proposed time, the negotiating will modify the proposal and choose a different time. The final output schedule returned after the negotiating process is unpredictable due to the many variables of the agents and the basic schedule.

3.4.3 Adaptation

During the scheduling procedure, the agents do not make decisions themselves. Instead, the solver adheres to the constraint provided by the traits of each agents. For example, the shower preference trait of an agent determines when an agent likes to shower. Therefore, the solver tries to schedule the shower task of this agent at a preferred time. However no direct agent behavior or adaptability exists during this process.

During the negotiation process, this agent adaptation does exist. The environment in this simulation exist of the household schedule. One negotiator agents tries to modify this schedule when a new event occurs. The responding agents adapt to this situation using their flexibility trait. When the negotiating agent tries to fit the event in the household schedule, it might require some modifications in the schedules of the other agents in the household. These changes involve reducing or relocating one or multiple tasks. Each agent is assigned a flexibility parameter, which indicates how much a task can be reduced.

This flexibility parameter changes depending on if the person is the requesting or responding agent. Requesting agents are assigned a higher flexibility value. It also depends on task type. For example, the shower task is the most flexible task, but the work task cannot be reduced and is therefore the least flexible.

Adaptions in this simulation are purely used to cater the requesting agent. They are social adaptions.

3.4.4 Objectives

The only objective during scheduling and negotiation is to find a solution that does not violate any constraints. These constraints can be no overlap or task order constraint. But this also includes constraints such as "a new shower cannot be scheduled further than a certain interval from the subtask". Another constraint is the flexibility of agents which only allows for a certain reduction of the task duration.

3.4.5 Learning

No learning aspects were implemented in this simulation.

3.4.6 Prediction

When agents are asked to respond to an event, they evaluate the impact that the proposed event will have on their schedule. They evaluate whether the event can fit in its current state or has to be modified. If modification is necessary, they send a counteroffer. The responding agents do not asses whether this counteroffer allows the requesting agent to fit. The responding agents also do not communicate with one other and therefore they cannot predict whether their proposed counteroffer will be sufficient to fit the proposed event.

3.4.7 Sensing

The negotiating agent does not sense the schedules of the responding agents directly. Instead, the agent asks the other agents for a response to his/her proposal. When agents are asked to respond to an event, they only evaluate their own schedule. They do not consider the schedules or counteroffers of the other agents. Therefore, all agents can only sense each others state variables through communication. However, this communication does not provide full insight in others' schedules.

3.4.8 Interaction

During scheduling, the entire process is handled by the solver and therefore agents do not communicate. During negotiation the requesting agent initiates communication with the other agents living in the same household. The requesting agent generates an event and creates a proposal to fit this event in the household schedule. Communication is initiated when the requesting agent asks the other agents to respond to the proposal. This request contains the information on the starting time and duration of the shower task the negotiator agent would like to schedule.

The responding agents respond by sending a reply. This reply contain a response code, either "Y" for accept or "N" for deny. If denied, the response contains a counteroffer which offers to slightly alter the responding agent's schedule. This offer contains a new starting time and duration for one of the responding agent's tasks.

The requesting agent waits until it has received a response from all agents. It then evaluates all responses. If all responses accept the proposal, changes are applied. Otherwise, the agent revises the proposal using the counteroffers and sends a new request to all agents.

3.4.9 Stochasticity

Each task has a random duration chosen from a certain interval/set of options. This variable was made stochastic to increase the variability in generated schedules. Average real-life duration of these tasks were unknown during development of this simulation. However, they are also not relevant because we are solely interested in how different types of schedules compare instead of the effect of the duration of specific task type. The main interest of this research, the showering practice, is based on data. It is still a stochastic variable but the duration is selected from a set of durations based on real-life data.

Another way to increase variability was the preferred shower time trait of each agent. This ensures that showers are not always planned in the most optimal timeslot. This stochastic variable and the random durations create schedules with different tightness, which allows us to test the effect of tightness on modification of the schedule.

Event generation is random. The event that is generated is random but also the person that generates the event is randomly selected. The person selection is limited to certain persons. For example, only full-time workers can generate early meeting events. This stochasticity is also used to increase variability among schedules. The cause of event generation was not relevant for this research. Only the effects of different types of events are measured during experimentation.

3.4.10 Collectives

The agents in the simulation are grouped in a collective called a Household. Only agents within the same Household can communicate and Households do not interact with or influence each other in any way.

Different types of agents were included in the simulation: Adults, Students, Children. All types share the same blueprint but the initialization of some of their variables is slightly different. For example, agents of type Children have a earlier bedtime. Also the set of daily tasks differs per agent type. For example, Students have a StudyTask while Adults have a full-time WorkTask.

The simulation also contains different types of tasks, e.g WorkTask or BedTimeTask. All blueprints for these task types are the same, but similarly to the agent types some variables are instantiated differently such as the release- and duedates and durations.

3.4.11 Observation

When the simulation is run, data is stored in two ways. The first data collection stores data in a dataset, which will be used for quantitative analysis. This dataset stores information for each household created in a separate entry. If the simulation generates 500 households, the dataset contains 500 rows of data. One entry contains information on the household's ID, score, tightness, negotiator tightness, event type etc. For exact specification see Section 5. The data for each household is only saved after a simulation run has been completed.

For the purpose of qualitative analysis, the schedules that are created during a simulation run are stored as pictures and text. Both the basic schedule and the modified schedule after negotiation are saved. The negotiation protocol that agents used to handle modifications to the schedule is also stored as text. This information shows what requests were made by the negotiator agent and what counteroffers were sent by the responding agents. The document also contains information regarding the occupants preferred shower time, their flexibility and the hType. This data is also only stored after the run was completed.

Simulation runs that generated a basic schedule which violated one or multiple hard constraints were excluded from the data collection. Therefore, if n=500 the actual n might slightly differ from this number. However, the percentage of schedules that were excluded from collection is relatively small. Further setup of data collection such as the variables that were varied during experimentation are further described in Section 5.

3.4.12 Code design

This subsection was originally not included in the ODD protocol. However some design decisions were made that should be explained but did not fit in the other subsections. These design decisions concern details of the model, particularly decisions based on code design.

Section 3.2 explained that tasks are not simply represented by one entity type. Besides the Task entity, a TaskAssignment entity was created to specify the starting times of tasks. The Task entities do not have this variable. But they are connected to a TaskAssignment entity which indirectly assigns a Task a starting time. This design choice was made due to the structure of the constraint satisfaction library. The solver requires a planning entity that stores the other planning attributes. This container entity allows us to connect a Task object to a TimeSlot object and store them in the same container. It also allows for extension of the model with other

planning variables. In the current model implementation, the duration of tasks is fixed. But using the TaskAssignment as a container object enables adding other planning variables, such as the duration. If multiple planning variables are present, the solver will assign both planning objects, such as the starting timeslot and the duration, to the tasks objects. However, both the starting time and durations have a rather large search space. Using both as planning variables would result in a long solving time. Therefore, the choice was made to use the duration as a fixed attribute.

Another type of entity used in the model is the TimeSlot entity, which is used to represent time. The solver requires the use of objects when assigning planning variables to the planning entity. Upon setup, a total of 96 TimeSlot objects are created and added to the Household's available timeslots list. We chose to create only 96 objects, regardless of the number of people living in the household. Each object represents a different 15 minute interval within the day. The first object is assigned a starting time of $00:00h$ and the last object has a starting time of $23:45h$. The 15 minute interval was chosen based on some experimentation. The options of 1, 5, 10 and 30 minutes were tested and compared to the 15 minutes interval. It was found that any interval smaller than 15 minutes tremendously slowed down the solving process, whereas any bigger intervals left some schedules unsolved as smaller gaps in the schedule were not considered by the solver. Therefore, the 15 minutes interval returns the most accurate solutions within an acceptable time-frame.

Instead of 96 objects in total, it was considered to generate 96 objects per person. But this was deemed redundant as every person within the household should have the same daily time available. Furthermore, using n times 96 objects would slow down the solving process tremendously, as the solver would have to consider $96 * n$ possibilities instead of just 96. The library also allows assigning the same planning variable object to multiple planning entities. However, this design choice can complicate extending the model because it allows multiple TaskAssignments to refer to the same TimeSlot object. Therefore, after solving was completed a new TimeSlot object was created for each TaskAssignment. The new TimeSlot objects were instantiated using the old TimeSlot objects. Thus, the new objects are identical to the old ones apart from the objects not being shared by multiple TaskAssignments. This eliminates the complications of cross-referencing.

The final design choice concerns the variety of subtasks in the model. A total of six subtasks is available for scheduling: RiseTask, BedTimeTask, WorkTask, StudyTask, DinnerTask and ShowerTask. An additional two subtasks were created for negotiation: SportsTask and PartyTask. It was considered to add several other tasks, however adding more tasks to the model is time expensive implementation-wise. Furthermore, it would further restrain the schedule which leads to longer solving times. If tighter schedules were required, the original objects' durations could be extended to cover for example travel time. It was found that these eight types of tasks filled the schedules enough but they are also generic enough to apply to an average person. The showering practice is the focus of this research, therefore this task had to be included. Work- and StudyTasks cover the majority of the day. Rise- and BedTimeTasks were added because every person requires sleep. They were separated to allow the schedule to range from $00:00 - 24:00$ hours. This way the time is represented as a monotone increasing number which eases the implementation.

3.5 Initialization

In our model we can define two states as the initial state. The first being the state of the Household upon creation and the second initial state occurs after scheduling has finished. In both scenarios we view the Household object as the model world, as this is the environment in which the agents live and negotiate. Upon Household creation all variables are created and initialized. But during the scheduling procedure the values of some of these variables are changed. Consequently, the initial world the negotiation phase receives differs from the world

upon creation in some aspects. We will describe both initial states below.

Household creation

A Household object is a collective of Person objects, TaskAssignments, Tasks and TimeSlots. As the pseudo-code in algorithm 2 indicates, all these objects are created and stored in separate lists in the generateHousehold method. The generation of the Tasks objects is not visible in this method, as it is hidden within the createPersons method.

Algorithm 2 Generate household

```
1: function createHousehold
2:     starting minute options ← every 15min of the day
3:     household ← new household
4:     household.persons ← createPersons()
5:     household.timeslots ← createTimeSlots(starting minute options)
6:     household.taskassignments ← createTaskAssignmentList()
7:     return household
```

In createPersons the Person objects are created and added to the Household. A random household is selected from a datafile that contains templates for Person objects. This datafile will be explained in detail in Section 3.6. For each person in the selected household in the datafile, a Person object is created. The data file contains a variable that describes the persons adolescence and studies. If the person is described as an adult but not as a student, an Adult object will be instantiated. Otherwise the Person object is instantiated as a Student object. If the person is not an adult, a Child object is created. The next variable that is instantiated is the job type. The data file also specifies this information. If the person is a full-time or part-time worker the job type is set to full-time or part-time respectively. Otherwise, the job type is set to unemployed. The final attribute read from the data is the persons student status. Since, both adults and children can be students this is not clear from the object instantization alone. The student status is stored in a boolean.

Two Person attributes are instantiated at random. These attributes concern the bedtime and risetime of a Person. The bedtime is set to a random time between 21h and 24h with steps of 15 minutes. If the Person concerns a child this range is changed to 20-22h. The risetime of all Persons is set to a random value in the interval 6h-11h, again with steps of 15 minutes. The final Person attribute is constant during the simulation and equal for all Person objects. The flexibility is set by the user using a slider in the experimental environment. The value ranges from 0.0 to 1.0, where 0.0 equals no flexibility and 1.0 is the maximum flexibility.

Following the object creation, the Persons are assigned a unique ID and the createTask method is called. This method creates a list of Tasks for that Person depending on the previously set variables. Each Person receives a ShowerTask, RiseTask, BedTimeTask and DinnerTask. If the agent is an Adult, a WorkTask is added. Similary, the student boolean specifiy the addition of a StudyTask. Each task is randomly assigned a duration. The duration ranges for each task type is preset in the corresponding subclass. The specific intervals can be found in Table 2.

Some of the preset durations are removed during task creation. This was done to assure that all tasks could still fit in the timespan of one day. If a Person works full-time all durations greater than 8 hours were removed for the RiseTask. Similarly, if a Person is a student all RiseTask durations above 9 hours were excluded. The job type determines the WorkTask duration. Full-time workers can only work 8 hours a day, while part-time workers work 4 hours a day. If the part-time worker concerns a child, the worktime is set to 1 hour a day. The duration of the BedTime is not variable, unlike Table 2 seems to suggest. The BedTimeTask indicates the time a Person spends sleeping before midnight, whereas a RiseTask indicates the sleeping time after midnight. During Person object creation the bedtime of each Person was initialized.

Therefore the starting time of the task is already known. The duration of the BedTimeTask should always be equal to 24h minus a Person's bedtime. E.g. if someone's bedtime is set to 22:30h, the BedTimeTask duration equals 90 minutes. The ShowerTask is the only task type not included in Table 2 because the durations for this task are drawn from a distribution. This distribution is drawn from a datafile (3.6) and depends on the meaning attribute. This meaning attribute is randomly assigned one of the following values: freshness, cleanliness or relaxation. Each meaning specifies a range of durations from which a random value will be drawn. This range is created by selecting all values from the data that fall in this interval. The ranges for each meaning are specified in Table 1.

Meaning	Min value	Max value
Freshness	1	15
Cleanliness	1	20
Relaxation	10	30

Table 1: Duration ranges for each ShowerTask meaning

Besides durations, release- and due-dates are preset in the task subclasses as well. These values determine the earliest time a Task can start and the latest time a Task can be completed respectively. The release- and due-dates are shown in Table 3 for each task type. The BedTimeTasks and RiseTasks both represent the sleeping tasks but they are instantiated in different ways. In the case of the BedTimeTask the bedtime variable determines when the task will commence. Contrarily, the risetime variable of a RiseTask does not regulate the starting time but the ending time or more specifically the duration of the Task. For this reason, the starting time of the RiseTask will always equal its releasedate while the ending time of the BedTimetTask will always equal its duedate.

The final preset Task attributes are the pushable and shrinkable booleans. These booleans indicate whether a task can be relocated or respectively reduced in duration during the negotiation phase. The instantiation for each task type is shown in Table 4. These instantiations are based upon intuition, e.g. a Party can be attended later but will not start later because one person cannot be in time. For the WorkTask, working hours are usually not very strict but the working time is fixed. Other tasks, such as the Rise- and BedTimeTask, are not pushable because this could cause gaps in the schedule during the night.

After Task object creation, they are assigned a unique ID within the Household and the complete list of Tasks of all occupants is added to the Household object. Thereafter, the Household is added as an attribute to all Person objects and a list of Person objects is stored as an attribute in the Household. Subsequently, the TimeSlots objects are created in the createTimeSlots method. This method is provided with all available starting minute options instantiated in the createHousehold method. For each possible starting minute option a TimeSlot object with a unique ID is created and the starting minute value is added as an attribute. This results in a total of 96 unique TimeSlots. The objects are stored in a list and added to the Household. We should note that TimeSlot objects are only created once, instead of once per person. The solver can use one TimeSlot object multiple times, unless it is restricted to do so. We only constrained the solver to use a TimeSlot object more than once per person. However, the solver is free to assign a TimeSlot object to different Persons simultaneously. The TaskAssignments are created in a similar way. For each Task previously created, a TaskAssignment object with the same ID is generated. Both the Person stored in the Task and the Task itself are added to the TaskAssignment entity and finally the list of all TaskAssignments is given to the Household. This list forms the unsolved schedule.

Task type	Shortest duration (minutes)	Longest duration (minutes)	Timestep (minutes)
Rise	300 (= 5 hours)	660 (= 11 hours)	15
BedTime	0	180 (= 3 hours)	15
Dinner	45	45	0
Work	60	480 (= 8 hours)	240
Study	360 (= 6 hours)	600 (= 10 hours)	30
Sports	30	90 (= 1.5 hours)	30
Party	60	150 (= 2.5 hours)	30

Table 2: Preset Task duration intervals

Task type	Releasedate (hours)	Duedate (hours)
Rise	00:00	12:00
BedTime	20:00	24:00
Dinner	17:00	21:00
Work	08:00	19:00
Study	06:00	24:00
Sports	08:00	22:00
Shower	06:00	24:00
Party	19:00	24:00

Table 3: Preset Task release- and due-dates

Initial schedules

During scheduling the Household objects have been changed in a few ways. The TaskAssignments stored in a Household were instantiated with empty TimeSlot objects, as the starting time is unknown during setup. The solver used constraint matching to find suitable timeslots for each of the assignment objects. During this process, TimeSlot objects are assigned to the starting timeslot attribute of each TaskAssignment. Therefore, the negotiation phase is presented with a fully initialized schedule.

Initially, Households only received a total of 96 TimeSlot objects, one for every 15 minutes of the day. After scheduling we replace this list with 96 TimeSlot entities for each person living in the Household. This prevents accidental modification of Tasks assigned to the same TimeSlot during negotiation. This design choice was further explained in Section 3.4. Every Person in the Household also receives his/her personal schedule. This schedule only contains the TaskAssignments that belong to the corresponding person and will be used during negotiation.

Before negotiation can start an Event is required. Event objects form the basis of the modifications made to the schedule. Depending on the type of Event that is generated, different values are assigned to its variables. First, a random eligible Person is selected to handle the negotiation of the Event. Which persons are eligible as well as several other variables depend on the event type. The specifications can be found in Table 5.

Three different types of events can be generated: earlyMeeting, eveningParty and workOut. The earlyMeeting event requires an advancement and extension of the WorkTask by 1 hour. The ShowerTask should be scheduled before the corresponding WorkTask. Similarly, the shower of the eveningParty event should also be scheduled before its subtask. This subtask is a PartyTask task which is added to the schedule as an extra evening task. The third type of event, the workOut event, requires the addition of a SportsTasks which should be scheduled after a work or study task. If the person has neither, the SportsTask is scheduled directly after the RiseTask. Contrary to the other events, the ShowerTask has to be scheduled after the SportsTask.

Task type	Shrinkable	Pushable
Shower	Yes	Yes
Work	No	Yes
Study	Yes	Yes
Rise	Yes	No
BedTime	Yes	No
Dinner	Yes	Yes
Sports	Yes	Yes
Party	Yes	No

Table 4: Overview of shrinkable and pushable tasks during negotiation

Some event traits are dependent on the event type. For example the time constraint, which is always equal to the start or end time of the subtask. Whether the time constraint is equal to the start or the end of the subtask depends on the before constraint. The max time to subtask trait does not depend on the event type and is always set to 2 hours, which implies that the Shower Task should take place within two hours of the start or end of the subtask. The requesting agent, ID of the changing TaskAssignment and duration of the Shower are also assigned during event generation. The proposed duration is a random number selected from the shower data. The interval from which the random number is selected, is restrained by the meaning of the shower. This meaning depends on the type of event that is generated. The earlyMeeting and eveningParty events create a shower with the purpose of freshness. The shower of the workOut event is used for cleanliness.

The number of attempts in the event is initialized at 0 and the adjusted TaskAssignments list is initialized as an empty list. The proposed starting time of the new ShowerTask is only assigned during negotiation. This time is dependent on the subtask of the Event, the first starting time that is proposed places the Shower directly before or after the subtask. The total reduction required is also assigned during the negotiation process. The differences among the event types are summarized in Table 5.

All Event types require a relocation or extra ShowerTask to be scheduled. While an Event is being generated, it is calculated whether the current ShowerTask in the negotiating agent's schedule is scheduled within a specific timespan from the subtask. When a shower is scheduled within this timespan, the current ShowerTask is added to the Event. This task will be relocated during the negotiation process. However, if the current ShowerTask exceeds this timespan an extra shower is added to the schedule. This extra ShowerTask is added to the Event and the original shower remains in its current position. The reason behind this are meanings attached to the shower practice, which are likely to change when shower occurs at a different time during the day. The earlyMeeting event requires a shower in the morning before the start of the WorkTask. If the original shower was scheduled after 14:00h a new ShowerTask object is added. The eveningParty event requires a shower in the evening, before the start of the PartyTask. A new ShowerTask is added when the original shower was scheduled more than 9 hours before the start of the PartyTask. The workOut event requires a shower after the SportsTask. The original shower is only relocated when it was scheduled within 2 hours after the SportsTask. Otherwise, an extra shower is added.

State variables

The simulation uses a few sliders which can be set by the user in the experiment environment. The nrHouseholds slider changes the number of Households created at once. This can be used to create a lot of instances at once, as results are written to files and can be analyzed and compared. Another slider sets the flexibility of the agents during negotiation. This slider ranges from 0.0

Event type	Subtask	Eligible persons	Shower placement	Shower meaning
earlyMeeting	None	Full-time workers	Before	Freshness
eveningParty	PartyTask	All	Before	Freshness
workOut	SportsTask	All but persons with both job and study	After	Cleanliness

Table 5: Initialization details for each Event type

to 1.0, where 0.0 is inflexible to change and 1.0 is maximal flexibility. The flexibility increase slider can be used to change the flexibility of the negotiating agent opposed to the responding agents. Ideally, the negotiator is somewhat more flexible than the responding agents as this agent is requesting alterations to the schedule.

3.6 Input data

Our model uses two datasets to instantiate some of its variables. These variables are the shower durations of the ShowerTasks and some of the attributes of the Person agents. Both datasets were extracted from data provided by Curtin University Sustainability Policy Institute (CUSP), Perth.

3.6.1 Household data

As explained in Section 3.5 some of the attributes of the Person agents are instantiated using data. This data was obtained by combining two datasets containing occupation information of several households in Fremantle near Perth, Western-Australia. The datasets described 10 and 14 different households, resulting in a sample size of 24. This data was combined and only the occupation details of the residents were used. This combined dataset contains the following information: number of occupants, occupation of each person (full-time, part-time or unemployed), age group of each person (adult or child) and student status. During Household object creation a random household is selected from this database and the Person objects are instantiated according to this data.

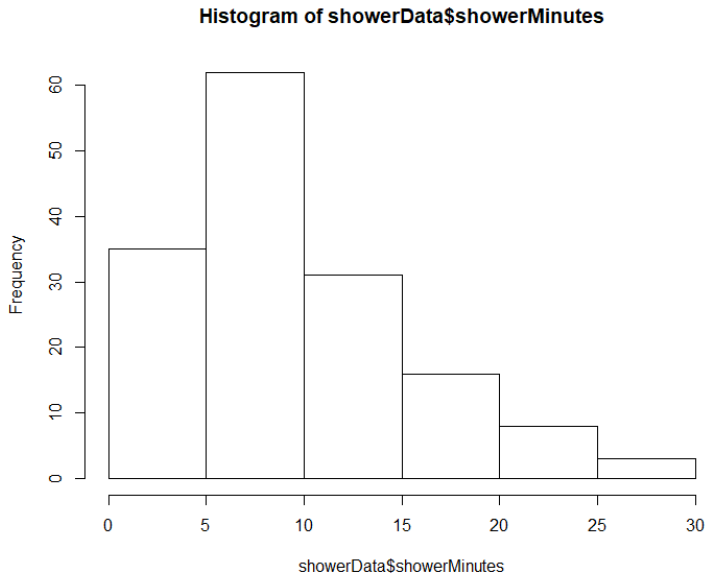


Figure 1: Shower duration distribution in minutes

3.6.2 Shower data

The durations of the ShowerTask are taken from a distribution. This distribution was created using multiple datasets provided by CUSP, Perth. For a total of 10 households the water, electricity, gas and other utility uses were recorded during a time period of 2 years. We selected 4 households from those 10 and sampled a total of 91 days from these remaining datasets. The other 6 households were not used because they contained some data disturbances or missing data, which complicates the analyzation process. In addition to this, some datasets required a different analysis process due to different hot water sources. From each of the remaining 4 datasets, 2 samples were randomly selected per month, which produces a total of 96 samples. Some households suffered from faulty recording systems during a specific period of time and therefore some months had to be excluded. The remaining 91 samples were used for analysis.

All 91 samples were analyzed manually. It was assumed that showers use 9 liters per minute and do not last longer than 30 minutes. The data was recorded in timeslots of 15 minutes, therefore a shower uses 135 liters at maximum each timeslot. If water use reached above this limit it was assigned to other appliances such as the washing machine, dishwasher or garden irrigation. Besides water use, we analyzed the hot water source use to distinguish garden irrigation from showers. Furthermore, it was assumed that the showers, dishwashers and washing machines were the only appliances using the hot water source. Households using the same source for hot water as central heating were filtered in the household selection step. It should be noted that the number of occupants in each household was known, including the persons' occupation. Only weekday data was used which allowed us to detect a pattern in the water use of each household. Every shower was marked in the datafiles, after which the durations of each individual shower was calculated. The histogram in Figure 1 shows the found distribution of shower durations.

3.7 Submodels

In this subsection we will discuss the methods mentioned in Section 3.3. All discussed Submodels are part of the negotiation phase of the simulation. Submodels related to the scheduling procedure will be discussed in Section 4.

The negotiation model was inspired by the PRINEGO model discussed in Mester et al. (2015). This paper provided pseudocode for their main negotiation method, which handles responses from the other agents and revision of the proposal. We extended this model by allowing the agents to respond with counteroffers if an event does not immediately fit in their schedule. The revision function of our model is more extensive than the revision done in PRINEGO, as revising schedules is more complicated than revising post offers. However, the selection of agents influenced by the offer is relatively easy as an Event always affects all members in the household.

Algorithm 3 Handles the negotiation of a generated event

```
1: function HANDLEEVENT(event)
2:     event ← eventPlacementSelf(event)
3:     if event ≠ null then
4:         return person.negotiate(event, c, m)
5:     else
6:         return null
```

As shown in the pseudo-code provided in Section 3.3 the negotiation phase starts by generating an Event. The generation of these Events has already been discussed in Section 3.5, therefore we will not discuss it here. Once an Event has been generated it is passed to the negotiator agent's handleEvent method. The pseudocode can be found in Algorithm 3. This

method performs a self-check in the agent's personal schedule before it initiates the negotiation process. The c and m parameters are the current number of iterations and the maximal number of iterations. They are used as an early stopping condition in the negotiate method.

Algorithm 4 Instantiate starting time of shower

```
1:  function eventPlacementSelf(event)
2:      nrAttempts ← event.nrAttempts
3:      event.nrAttempts ← nrAttempts + 1
4:      event.duration ← event.duration + event.reduction
5:      event.reduction ← 0
6:      event.adjustedtasks ← ∅
7:      if event.beforeconstraint then
8:          event.starttime ← event.timeconstraint − event.duration ∗ (nrAttempts + 1)
9:      else
10:         event.starttime ← event.timeconstraint + event.duration ∗ (nrAttempts)
11:     eventStart ← event.starttime
12:     eventEnd ← eventStart + event.duration
13:     if event.beforeconstraint then
14:         if event.timeconstraint − eventEnd ≥ event.maxtimetosubtask then
15:             newEvent ← null
16:             if event.subtask ≠ null then
17:                 newEvent ← relocateSubTask()
18:             if newEvent == null AND !event.othertaskpushed then
19:                 event.othertaskpushed ← true
20:                 newEvent ← pushOtherTasksSelf(event, false)
21:                 if newEvent == null then
22:                     return pushOtherTasksSelf(event, true)
23:             return newEvent
24:     else
25:         if eventStart − e.timeconstraint ≥ event.maxtimetosubtask then
26:             newEvent ← null
27:             if event.subtask ≠ null then
28:                 newEvent ← relocateSubTask()
29:             if newEvent == null AND !event.othertaskpushed then
30:                 event.othertaskpushed ← true
31:                 newEvent ← pushOtherTasksSelf(event, false)
32:                 if newEvent == null then
33:                     return pushOtherTasksSelf(event, true)
34:             return newEvent
35:     if selfCheck(event) then
36:         return event
37:     else
38:         return eventPlacementSelf(event)
```

In Algorithm 4 the number of attempts is increased every time this function is called and the event variables are reset. It is checked whether we are dealing with a before or after event. This indicates whether the requested shower should be added before or after the subtask. The first assignment of the start time places the shower task directly before or after the subtask. If for some reason the negotiation fails, the shower is moved a little further from the subtask if it does not exceed the maximum time range. If a possible start time was found, the agent tries

to fit the event in its schedule using selfCheck. However, if no suitable timeslot could be found for the shower task, it is tried to relocate the subtask. This subtask relocation only works for the eveningParty and workOut event as the earlyMeeting event does not have a subtask. If the subtask could not be relocated, it is tried to push the other tasks of the requesting agent. Pushing the other tasks is not tried when the event is already in the process of pushing.

The relocation of the subtask is handled by Algorithm 5. The new start of the subtask is equal to its current starting time plus the maxtime to subtask. This way no possible shower slots are skipped because each relocation skip is equal to the max time to subtask. The algorithm checks whether the subtask can move to the skipped starting time. If it overlaps with any of the agent's other tasks, the subtask is relocated to the end of the proposed time. This way relocation is tried in small steps and possible fitting gaps cannot be missed. When relocation is successful, the while loops ends. The algorithm checks whether the subtask has not been moved outside the daily hours and returns the updated event.

Algorithm 5 Relocate the subtask of an event

```
1:  function RELOCATESUBTASK(event)
2:      mySchedule ← event.requestingagent.schedule
3:      newstart ← substart + event.maxtimetosubtask
4:      restart ← true
5:      while restart do
6:          restart ← false
7:          newend ← newstart + subtask.duration
8:          for each taskassignment in mySchedule do
9:              if !(taskass.end ≤ newstart OR newend ≤ taskass.start) then
10:                 restart ← true
11:                 newstart ← taskass.end
12:     if newstart ≥ 24 ∗ 60 then
13:         return null
14:     event.subtask ← subtask
15:     return event
```

In Algorithm 6 it is tried to push the other tasks in the requesting agent's schedule to fit the subtask. Only tasks that block the window in which the shower could be scheduled are pushed. This window is the start minus the maxtime to subtask or the end plus the max time to subtask, depending on the beforeconstraint. It is checked for every taskassignment in the agent's schedule whether it was scheduled (partially) during this time window. The shower task itself is excluded from the check, as this task is the one being rescheduled. The task should be pushable if we want move it. For each task that was found to overlap with the time window, the previous and next task in the schedule are calculated. The time difference between the end of the previous task and the beginning of the overlapping task indicates the left push distance. The time difference between the start of the next task and the end of the overlapping task determines the right push distance. These are the maximum distances the overlapping task can be pushed to the left and the right respectively.

It is checked whether the overlap between the task and the time window occurs on the left or right side of the time window. If the overlap occurs in the right part, pushing to the left is tried first. Otherwise, the task is pushed to the right. The maximum and required push distances are calculated. The minimum of the two distances is subtracted from the starting time of the overlapping task. This forms the new starting time for the overlapping time. Hereafter, it is checked whether reduction of tasks is allowed. The first call of this method tries fit the shower task without reducing any task's duration. However, when a second call is required this implies that simply pushing tasks was not sufficient and reduction is allowed. If the overlapping

task is shrinkable, it is calculated whether the required distance is bigger than the maximum pushing distance. The algorithm calculates the maximum shrinkage of the respective task, this depends on the task type and the agent's flexibility. The task's duration is only reduced by the maximum required and allowed shrinkage. The shrinkage always occurs on the opposite side as the pushing. In other words, if a tasked is pushed to the left, the task can only be reduced on the right side and vice versa.

A new Event object is created with the pushed and potentially shrunk task details. Thereafter, the handleEvent method (Algorithm 3) is called to continue the negotiation process. If the negotiation with the updated pushed and shrunk task still fails, the task is reset to its original position and it is tried to push in the opposite direction.

In Algorithm 5 the function *getFlexibility* is called. This function determines the flexibility of the agent with respect to the type of task that is being shrunk. Some tasks are more flexible than others, this is reflected by subtracting some value from the original agent flexibility. The flexibility subtractions for each task can be found in Table 6. A maximum function is used to assure that the flexibility does not become negative.

Algorithm 7 shows the calculation used to reduce the task duration. The minimum duration is dependent on the current duration of the task and the agent's flexibility. For example, suppose an agent has a flexibility value of 0.5 and the current task duration is equal to 10 minutes. The minimum time requested by this agent is 7.5 minutes, which is rounded down to 7 minutes. If the minimum requested time violates the minimum duration threshold, the minimum duration will be set to 2 minutes.

In Algorithm 8 the agent checks for overlap with the event in its personal schedule. It examines whether the overlap occurs on the left or right side of the new shower task. It calculates the number of minutes its willing to subtract from its current shower duration using its flexibility parameter. This flexibility parameter is dependent on the meaning of the shower. Some meanings reduce or increase the flexibility of the agent, as shown in Table 7. The flexibility never exceeds the $0 - 1$ interval.

Task type	Flexibility
ShowerTask	$agentflex$
WorkTask	0
StudyTask	$agentflex - 0.3$
DinnerTask	$agentflex - 0.1$
BedTimeTask	$agentflex - 0.2$
RiseTask	$agentflex - 0.2$
PartyTask	$agentflex - 0.1$
SportsTask	$agentflex - 0.1$

Table 6: Flexibility of each task type, where $agentflex$ is the flexibility of the agent

Shower meaning	Flexibility
Cleanliness	$agentflex$
Freshness	$agentflex + 0.3$
Relaxation	$agentflex - 0.3$

Table 7: Flexibility of each shower meaning, where $agentflex$ is the flexibility of the agent

Algorithm 6 Relocate other tasks of the negotiating agent

```
1: function pushOtherTasksSelf(event, allowReduce)
2:     if event.beforeconstraint then
3:         windowEnd ← event.timeconstraint
4:         windowStart ← windowEnd − event.maxtimetosubtask − event.duration
5:     else
6:         windowStart ← event.timeconstraint
7:         windowEnd ← windowStart + event.maxtimetosubtask
8:     for each task in personal schedule do
9:         if task.id ≠ event.id then
10:            if task.pushable then
11:                overlap ← min(taskend, windowEnd) − max(taskstart, windowStart)
12:                if overlap > 0 then
13:                    endprevtask ← −1
14:                    startnexttask ← max integer value
15:                    for each othertask in personal schedule do
16:                        if othertask.id ≠ event.id then
17:                            if othertask.end ≤ task.start AND
                                   othertask.end > endprevtask then
18:                                endprevtask ← othertask.end
19:                            if othertask.start ≥ task.end AND
                                   othertask.start < startnexttask then
20:                                startnexttask ← othertask.start
21:                    pushleft ← ||task.end − windowStart|| < ||task.start − windowEnd||
22:                    if pushleft then
23:                        leftdist ← task.start − endprevtask
24:                        requireddist ← overlap
25:                        pushdist ← min(leftdist, requireddist)
26:                        newstart ← task.start − pushdist
27:                        task.start ← newstart
28:                        dist ← 0
29:                        if allowReduce then
30:                            if task.shrinkable then
31:                                diff ← requireddist − leftdist
32:                                if diff > 0 then
33:                                    taskflex ← rulebase.getFlexibility(task, person.flexibility)
34:                                    maxshrink ← task.duration−
                                           rulebase.getMinDuration(taskflex, task.duration)
35:                                    diff ← min(maxshrink, diff)
36:                                    task.duration ← task.duration − diff
37:                        newEvent ← person.handleEvent(event)
38:                        if newEvent ≠ null then
39:                            return newEvent
40:                        task.start ← originalstart
41:                        task.duration ← task.duration+ max(0, diff)
42:                        if requireddist > leftdist + diff then
43:                            push right, similar to left push
44:                    else
45:                        push right, similar to left push
46:    return null
```

Algorithm 7 Calculate the minimal task duration given the agents flexibility

```
1: function getMinDuration(flexibility, duration)
2:     minDuration ← 2
3:     reduction ← −duration/2 ∗ flexibility + duration
4:     if reduction < minDuration then
5:         reduction ← minDuration
6:     return floor(reduction)
```

Algorithm 8 Try to fit event in personal schedule

```
1: function selfCheck(event)
2:     reductionL ← 0
3:     reductionR ← 0
4:     flex ← ruleBase.getShowerMeaningFlexibility(evening.meaning, person.flexibility)
5:     minShowerTime ← getMinDuration(max(flexibility + increase), event.duration)
6:     maxReduction ← event.duration − minShowerTime
7:     for t in personal schedule do
8:         if event.taskID ≠ t.ID then
9:             if t overlaps with event then
10:                if event.end < t.start then
11:                    reductionR ← reductionR + overlapinminutes
12:                else
13:                    reductionL ← reductionL + overlapinminutes
14:    if reductionL + reductionR < maxReduction then
15:        updateEvent(event, reductionL, reductionL + reductionR)
16:        return true
17:    else
18:        return false
```

Algorithm 9 Push and reduce the time of the event

```
1: function updateEvent(event, pushdistance, reduction)
2:     if pushdistance < 0 then
3:         pushdistance ← pushdistance + reduction
4:     event.duration ← event.duration − reduction
5:     event.reduction ← event.reduction + reduction
6:     event.starttime ← event.starttime + pushdistance
```

If the reduction necessary to fit the event in the schedule does not exceed the maximum reduction, the event is updated using Algorithm 9. When a task is pushed, the starting time of the event is changed by adding or subtracting a time difference. This moves the task to the right or respectively left in the schedule. The location of a task in the schedule is dependent on its starting time and duration. When a duration reduction on the right side of a task is required, the duration of the task can simply be reduced. However, if a reduction on the left side is required, the task should be pushed towards the right as well.

After the requesting agent has analyzed its personal schedule and found a suitable timeslot for the event, the agent initializes the negotiation process. The negotiation method presented in Algorithm 10 first checks the early stopping condition. If the stopping condition has not been met yet, all other occupants are asked for a response to the event proposed by the requesting agent. If all agents send the response code "Y", the event is returned and negotiation was

successful. If at least one of the agents send the response code "N", the counteroffers are inspected. If the counteroffers allow the event to fit in the schedule, the event is returned and negotiation is successful. Otherwise, the event has to be revised by the negotiator agent. If an alternate event can be generated, the negotiation process is restarted. If not, negotiation was unsuccessful.

Algorithm 10 Send offers to other agents and check if their counteroffers are sufficient. Otherwise revise the offer and renegotiate

```
1: function NEGOTIATE(event, current try, max try)
2:     if current try > max try then
3:         return event
4:     responses ← ∅
5:     occupants ← getPersonList()
6:     for person in occupants do
7:         if person ≠ self then
8:             response ← person.ask(event)
9:             responses.add(response)
10:    if responses.All("Y") then
11:        return event
12:    else
13:        if checkCounterOffer(event, responses) then
14:            acceptCounterOffer(event, responses)
15:            return event
16:        alternate event ← revise(event, current try, max try, responses)
17:        if alternate event ≠ null then
18:            return negotiate(alternate event, current try, max try, responses)
19:        else
20:            return null
```

Algorithm 11 Call the rule base of the responding agent

```
1: function ASK(event)
2:     response ← agent.checkRuleBase(event)
3:     return response
```

Algorithm 11 calls the rule base of an agent when asked to respond to a proposal. This rule-base, presented in Algorithm 12, will check for any contradictions. It is checked whether the requested shower overlaps with the responding agent's shower. If no overlap was found, the response "Y" is sent. Otherwise, it is calculated whether the overlap occurs mostly on the left or right side of the requested shower. The responding agent generates a counteroffer that proposes a duration reduction of its task on the overlapping side. The pseudo-code for the rule base can be found in Algorithm 12.

Once the requesting agent has received all responses, the counteroffers are inspected one by one by Algorithm 13. If the counteroffers were not sufficient to resolve all conflicts, the check fails and the event will have to be revised. Otherwise, the counteroffers will be processed and accepted using Algorithm 14. The adjustments made to the existing tasks in the schedule are stored in the event.

Algorithm 12 Rule-base of an agent

```
1: function checkRuleBase(agent, event)
2:     shower ← agent.schedule.getShower()
3:     response.agent ← agent
4:     if no shower in personal schedule then
5:         response.answer ← "Y"
6:     else if event shower overlaps personal shower then
7:         if ||shower end − event start|| < ||shower start − event end|| then
8:             response.answer ← "N"
9:             flex ← getShowerMeaningFlexibility(meaning, agent.flexibility)
10:            minShowerTime ← getMinShowerTime(flex, showerduration)
11:            reduction ← showerduration − minShowerTime
12:            counteroffer.starttime ← showerstart
13:            counteroffer.endtime ← showerend − reduction
14:            counteroffer.taskID ← shower.ID
15:            response.counteroffer ← counteroffer
16:        else
17:            response.answer ← "N"
18:            flex ← getShowerMeaningFlexibility(meaning, agent.flexibility)
19:            minShowerTime ← getMinShowerTime(flex, showerduration)
20:            reduction ← showerduration − minShowerTime
21:            counteroffer.starttime ← showerstart + reduction
22:            counteroffer.endtime ← showerend
23:            counteroffer.taskID ← shower.ID
24:            response.counteroffer ← counteroffer
25:    else
26:        response.answer ← "Y"
27:    return response
```

Algorithm 13 Check if the offer fits given the counteroffers

```
1: function checkCounterOffer(event, responses)
2:     for response in responses do
3:         offer ← response.getCounterOffer()
4:         if offer ≠ null then
5:             if offer overlaps with event then
6:                 return false
7:     return true
```

Algorithm 14 Accept counteroffers by storing changing in event

```
1: function acceptCounterOffers(event, responses)
2:     counteroffers ← event.adjustedtaskassignments
3:     for r in responses do
4:         offer ← r.counteroffer
5:         if offer ≠ null then
6:             counteroffers.add(offer)
7:     event.adjustedtaskassignments ← counteroffers
```

Algorithm 15 Revises offer based on received counteroffers

```
1: function REVISE(event, current try, max try, responses)
2:     for response in responses do
3:         offer ← response.getCounterOffer()
4:         if offer ≠ null then
5:             if offer overlaps with event then
6:                 pushLeft ← ||offerStart − eventEnd||
7:                 pushRight ← ||offerEnd − eventStart||
8:                 if pushRight < pushLeft then
9:                     requiredReduction ←tryFit(event, pushRight, responses)
10:                    if requiredReduction ≥ 0 then
11:                        acceptCounterOffer(event, responses)
12:                        updateEvent(event, pushRight, requiredReduction)
13:                        return event
14:                else
15:                    requireReduction ←tryFit(event, -pushLeft, responses)
16:                    if requireReduction ≥ 0 then
17:                        acceptCounterOffer(event, responses)
18:                        updateEvent(event, −pushLeft, requiredReduction)
19:                        return event
20:    return eventPlacementSelf(event)
```

Algorithm 16 Try to fit the event by pushing and reducing given the counteroffers

```
1: function TRYFIT(event, pushDistance, responses)
2:     pushedStart ← eventStart + pushDistance
3:     pushedEnd ← pushedStart + event.duration
4:     requiredReduction ← 0
5:     flex ← ruleBase.getShowerMeaningFlexibility(event.meaning, agent.flexibility)
6:     minShowerTime ← getMinShowerTime(self.flexibility, event.duration)
7:     maxReduction ← event.duration − minShowerTime
8:     for taskAssignment in requestingAgent.schedule do
9:         if event.taskID ≠ taskAss.ID then
10:            overlap ← min (pushedEnd, taskEnd) − max (pushedStart, taskStart)
11:            if overlap > requiredReduction then
12:                requiredReduction ← overlap
13:            if requiredReduction + event.currentReduction > maxReduction then
14:                return −1
15:    for response in responses do
16:        offer ← response.offer
17:        if offer ≠ null then
18:            overlap ← min (pushedEnd, offerEnd) − max (pushedStart, offerStart)
19:            if overlap > requiredReduction then
20:                requiredReduction ← overlap
21:            if requiredReduction + event.currentReduction > maxReduction then
22:                return −1
23:    return requiredReduction
```

If the proposed counteroffers were not sufficient, the event is revised using the revise method shown in Algorithm 15. The counteroffers are taking into consideration while trying to relocate

the event task in the schedule. It is also tried to reduce the duration of the requested event. The tryFit method, shown in Algorithm 16, calculates the required reduction. The pushdistance provided by the revise method moves the event task slightly to the left or right in the schedule. Therefore, the tryFit method inspects if the pushing has introduced any new overlap in the personal schedule of the requesting agent. If this is the case, it is checked whether the task's duration can be reduced to prevent this collision. The maximum reduction is calculated using the agent's flexibility, which equals the default flexibility value plus the flexibility increase because it concerns the requesting agent. When the required reduction exceeds the maximum reduction, the pushing has caused an impossible fit in the agent's personal schedule. If the reduction does not violate this threshold, it is reassessed whether the counteroffers still overlap with the event task. This overlap can be resolved by further reducing the event task's duration. If a positive reduction value is returned to the revise method, the reduction was sufficient to fit the event task in the schedule. Subsequently, the counteroffers are accepted and the event is updated with the necessary relocation and reduction. If the changes offered by the responding and requesting agents were not sufficient, Algorithm 4 is called. This method increases the attempt counter, relocates the event and restarts the event handling process.

Algorithm 17 Modifies the schedule to fit the event

```
1:  function applyChanges(requesting person, household, event)
2:      if event == null then
3:          for t in household.taskassignments do
4:              if t.task instanceof ShowerTask then
5:                  if t.start == null then
6:                      remove empty shower object
7:                  else
8:                      for t2 in household.taskassignments do
9:                          if t ≠ t2 then
10:                             if t.start ≠ null then
11:                                 if t overlaps with t2 then
12:                                     remove t from schedule
13:         return household
14:     for t in household.taskassignments do
15:         if t.ID == event.taskID then
16:             t.startingtime ← event.startingtime
17:             t.duration ← event.duration
18:     if event.subtask ≠ null then
19:         substart ← event.subtask.start
20:         subID ← subtask.id
21:     for counteroffer in event.adjustedtaskassignments do
22:         for t in household.taskassignments do
23:             if counteroffer.ID == t.ID then
24:                 t.startingtime ← counteroffer.startingtime
25:                 t.duration ← counteroffer.duration
26:             if subID == t.ID then
27:                 t.startingtime ← substart
28:     return household
```

Once the event handling process has finished, the applyChanges method is called to modify the existing schedule. Algorithm 17 shows the pseudo-code of this method. When an event was not negotiated successfully, an empty event is returned. This means no suitable time for the requesting agent's shower task could be found and has to be removed from the schedule. The

changes made by event's subtask, such as the early meeting event which extends the working duration, were already processed during the event generation step. These changes are not removed from the schedule because they are independent of the agent's showering task. An early meeting is not canceled when an agent cannot find a suitable shower time in advance. The new schedule is returned. If the event was negotiated successfully, the starting time and duration of the original shower task of the requesting agent are updated in the Household's schedule. Subsequently, all TaskAssignments influenced by the event are updated by processing the counteroffers stored in the adjusted TaskAssignment list. The new schedule is returned.

4 Scheduling

Our model combines the concept of scheduling with the concept of agent-based modeling. The scheduler is responsible for creating a basic schedule, while the agent-based model allows for modifications within this schedule. A household has one or multiple occupants who perform daily tasks. These tasks need to be performed in a specific order or at specific times. The initial order of these tasks is decided by an external mechanism, called a scheduler. In order to create this initial schedule, the scheduler resorts to constraints. These constraints depend on the social context such as socially accepted work times or bathroom availability. The preferences of occupants can also be expressed in constraints, e.g. a preferred shower time. These preferences are different for each occupant and are used to express the self-identity of the occupant. Occupants can also attach meaning to certain tasks, such as taking a shower to relax, freshening up before work or get clean. This meaning is part of the self-identity of the occupant. If multiple occupants wish to shower at the same time, the scheduler will not allow this as the shower is a limited resource.

The final schedule will incorporate all these constraints and preferences. It combines the occupant logic within one entity, the household. The schedule forms a basic order in which daily tasks are executed. This conforms with the psychological definition of habits. Behavior is often the cause of a goal that an individual wishes to achieve. But after many repetitions of the same behavior it becomes a habit and the individual no longer needs to bear the goal in mind. In our model these habits are created by scheduling the tasks of a household at the start of the simulation. This basic schedule is used as the start of negotiation within the household. Individuals do not have to actively consider each and every one of their actions, but can submit to their automatic behavior. Once the basic schedule is finished and negotiation can be initiated, the external mechanism is no longer in control. The fundamentals of the scheduling and the constraints used to create the basic schedule are explained in the following two subsections.

4.1 The fundamentals of scheduling

Scheduling is a theory within computer science which concerns itself with assigning specified tasks to resources. In our model every household has a number of occupants and each occupant has daily activities. These daily activities need to be scheduled alongside plans of other occupants. Each activity, hereafter called task, has several constraints that need to be met. For example, the dinner task can only be scheduled after work and before the sleeping task. Furthermore, most families prefer to have dinner together and therefore the dinner tasks of all occupants are preferably planned at the same time.

Scheduling theory deals with allocation of (scarce) resources to tasks over time. Its goal is to optimize the schedule given one or more objectives (Pinedo, 2016). Such planning problems are found in many different situations, such as manufacturing companies, vehicle routing or course timetabling. The resources and tasks take different forms in each situation. Resources can be machines in a factory, employees at a company or processing units in a computer. Tasks can be production processes, meetings or computer programs. The objectives can also differ in each application. Some objectives try to minimize the total processing time of all jobs in the schedule. Other objectives may be maximizing profit or happiness of employees and customers.

In this paper we will focus on finite and deterministic scheduling models. Finite models assume that the number of jobs that need to be scheduled and the number of machines are finite. A deterministic model only deals with planning problems of which the job data is fully known. In stochastic models processing times, release dates or due dates may not be known in advance. Usually a distribution is known however, the exact values only become apparent once a job has finished processing. In our model all job data is generated and set before handed to the scheduler.

Before going in depth on the details of our scheduling model, the basics of scheduling will be

discussed. The used notation is based on the notation in Pinedo (2016). In scheduling problems jobs have to be allocated to resources over time. These resources are often called machines and the number of machines is denoted by m. The number of jobs is denoted by n. Subscripts are used to refer to a specific job or machine. For jobs the subscript j is used and machines use the subscript i. The pair (i, j) refers to job j being processed on machine i. All jobs in the model have a processing time, p_{ij}. This is the time it takes to complete job j on machine i. In some cases jobs have a release date r_j and/or a due date d_j. These dates indicate the timespan in-between which job j has to be completed. A job j cannot start processing before the release date and a penalty applies when a job finishes processing after its due date. If the due date *must* be met, this is called a deadline. Some jobs are more important than other jobs. This can be indicated by giving jobs different weights, denoted by w_j.

A planning problem is described by the triplet $\alpha|\beta|\gamma$. α describes the machine environment, β describes the processing characteristics and γ describes the optimization criteria. Schedules can have many characteristics, since the scheduling theory applies to a wide variety of situations. We will explain only the characteristics necessary to understand our model. In the α field the number and types of machines is described. In our model the occupants of the households are considered the machines. Each household has 1 to 5 machines. The machines are not identical because occupants have different characteristics. In the case of multiple occupants, the machines are in parallel. The speeds of the same type of job can vary depending on the machine. E.g. one person can sleep for 6 hours while another person sleeps for 8 hours.

In the β field the processing restrictions and constraints are specified. Our model uses release dates for the tasks, such as the dinner task. The use of due dates is not specified in the scheduling triplet. The optimization criteria will indirectly imply the existence of due dates. Some jobs in the system have precedence constraints. These type of constraints imply that one or more jobs have to be completed before job j is allowed to start processing. E.g. a person has to get out of bed before starting any other tasks. Another characteristic is the machine eligibility restrictions. This implies not all machines are capable of processing each type of job. As an example, in our model each occupant has its own task list. Therefore, the tasks in this list can only be completed by the owner of the list.

In this paper a few assumptions are made which simplifies the scheduling process. It is assumed that once a job has started processing, it cannot be interrupted until completion. This allows us to simplify the model and reduce the search space for the solver. It is also assumed that the jobs in our simulation do not require any setup time in-between tasks. Meaning, one tasks can be directly planned after completion of another task. Travel times are also not included. If necessary, these can be incorporated by extending the job processing time. Finally, the model assumes no batch processing, a machine can process only one job at a time. The optimization criteria in the γ field will be explained later.

After all machines and jobs are created and specified, the scheduler has to solve the schedule. Solving means that the jobs are allocated to the machines while optimizing the objectives. There are many different algorithms for assigning these jobs to the machines, these are called optimization algorithms. Which optimization algorithm is used depends on the situation. However, optimization is often a trade-off with duration. In real world scheduling problems the search space is often very large. In these cases it is often impossible to find the optimal solution in the available time. Therefore, a trade-off is made and a solution is accepted once a stopping condition is met. E.g. when the score exceeds a certain threshold or when time runs out.

4.2 The constraint satisfaction solver

In this section we will explain the scheduling phase of the model in detail. This phase is used to create basic schedules for the households in the simulation. All TaskAssignments within a Household are assigned an appropriate starting time while adhering to several hard and soft constraint. These constraints will also be described in detail in this section. The entire

simulation was made in Java. For the purpose of scheduling we used the Optaplanner[1] library, which is a constraint satisfaction engine that optimizes planning problems and can be easily embedded in existing software.

Name	Hard/Soft	Score	Description
noTaskOverlapWithinPersonalSchedule	Hard	-1	Tasks within each agent's personal schedule cannot overlap.
showerScarceResource	Hard	-1	ShowerTasks cannot overlap
invalidStartingOrEndingTime	Hard	-1	Tasks cannot start before their releasedate and must end before their duedate.
invalidTaskOrder	Hard	-1	In the personal schedule of agents, tasks cannot start before their preceding tasks have finished.
durationEqualsMidnight	Hard	-1	Starting time of a BedTimeTask must equal midnight minus the duration of the task.
riseFromMidnight	Hard	-1	The starting minute of a RiseTask must always equal 0.
familyDinner	Soft	-10	Preferably everyone in the same household has dinner at the same time.
showerTimePreference	Soft	-2	Preferably a persons shower is scheduled according to his preference.

Table 8: Hard and soft constraints used in during the scheduling phase

The scheduling process is initiated by generating and instantiating a solver factory object. This factory object is created using the planner configuration as specified in an XML file. The file specifies which Java project contains the planning entities, solutions and variables. The classes in the Java project have been annotated to indicate which objects need to be scheduled. The annotations of the classes can be found in Appendix A.

The XML file also specifies the optimization criteria and the file in which the scoring rules can be found. These optimization criteria serve as a stopping rule for the solver. Once one of these stopping conditions has been met, the solver will abort and the found solution will be returned. For the purpose of this simulation and the conducted experiments, we set two stopping conditions. When a planning solution has been found with a score of $0hard/0soft$, no constraints are violated and an optimal solution has been found. If the solving process takes a long time, the scheduling process is aborted early on. After some testing, we set this second stopping condition to 20 seconds in real time.

The scoring rule file contains all hard and soft constraints to which the solver should adhere when trying to find a satisfactory solution to the planning problem. The initializing scoring trend was set to only minimize the score because only negative constraints were used. We use two levels of constraints: hard constraints and soft constraints. Both constraint types are negative, meaning a penalty is imposed when the constraint is broken. A hard constraint must not be broken, for example the shower cannot be used by two people at the same time. While

[1] https://www.optaplanner.org/

a soft constraint should not be broken if it can be avoided, e.g. families like to have dinner together. All constraints used during the scheduling phase are shown in Table 8.

Once a schedule has been solved, a basic schedule is given as output. This basic schedule forms a solution to the planning problem. All taskassignments in the household were assigned a timeslot which adhere to the constraints. An example of a solution is shown in Figure 1. The score of a optimal solution equals $0hard/0soft$. In this case no hard or soft constraints are violated in the provided solution. If a hard constraint is violated a penalty of -1 is imposed. This penalty is the same for all hard constraints. On the contrary, soft constraints impose different penalties when violated. The scores can be found in Table 8. The soft constraints were assigned different penalty values to express the importance of each rule.

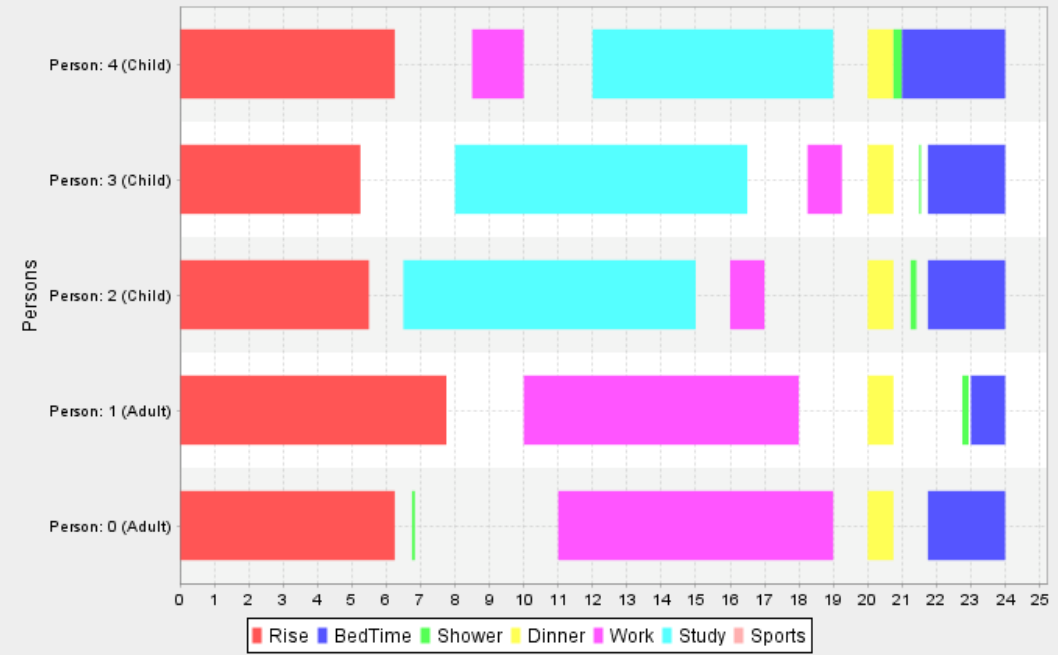


Figure 2: A basic schedule solution of a household with 5 occupants

We will briefly discuss the meaning and usefulness of each constraint provided in Table 8.

Each Person within the household has his/her own set of tasks to complete during the day. Logically, only one task can be executed at the same time. Therefore the next task can only start after the previous task has been completed. Several constraints were implemented to enforce a certain order in which the tasks are scheduled. Most importantly, two tasks cannot be scheduled at the same time. When (partial) overlap occurs within the schedule, the *noTaskOverlapWithinPersonalSchedule* constraint will penalize this. This constraints prevents overlap among tasks in each agent's personal schedule. A second constraint that restricts the order in which the tasks can be scheduled is imposed by the release- and duedates that were assigned to the tasks during setup. These dates are times of the day and are preset for each task type as shown in Table 3. The constraint is violated when a task is scheduled before its releasedate or is completed after its duedate. If violated a penalty will be imposed by the *invalidStartingOrEndingTime* constraint.

A final constraint that restricts the task order is the *invalidTaskOrder* constraint. Some tasks need to be completed before other tasks. For example, the first task of the day should always be the RiseTask. An agent cannot start any other tasks before getting out of bed first. The bedTimeTask is always the last task in an agents schedule. Details on all preceding and succeeding tasks can be found in Table 9. The SportsTask and PartyTasks do not appear in this table, because these tasks do not exist during the schedule procedure. They only become available during event generation and the negotiation phase.

Specific to this simulation are the shower tasks. The bathroom is a scarce resource and each household is assumed to only have one shower facility available at a time. The *showerScarceResource* constraint was implemented to ensure that the shower is treated as a scarce resource. Only one person within the household can shower at a time. In other words, this rule ensures that two ShowerTasks cannot overlap. All four order enforcing constraints are implemented as hard constraints. Schedules that violate any of these constraints will not be considered for experimentation.

Constraints that should be optimized but are allowed to be violated are called soft constraints. We implemented two social rules as soft constraints. The first rule is the *familyDinner* constraint. This soft constraint induces a preference for having dinner together with the rest of the family. This family consists of all agents living in the same household. This rule also ensures a central point of the day for all agents.

The second social rule is the *showerTimePreference* constraint. Every agent in the household has a preferred time of the day to shower. If this soft constraint is not violated every agent's shower is scheduled during it's preferred time period. The shower preference can be set to morning (06:00-11:00h), evening (20:00-24:00h) or no preference. This constraint was added to increase variation within schedules and to ensure that schedules are not always solved in the optimal way. For example, an agent might have a lot more time to shower in the morning, but prefers to shower in the evening. This causes the schedule to be tighter.

The two final constraints do not enforce any task order or social rules. They were implemented to ensure a working scheduler system. The Rise- and BedTimeTask are two special task types, as one of their variables is already fixed. The RiseTask has a fixed starting time while the BedTimeTask has a fixed ending time. The scheduler does not know that these variables should be fixed and therefore the *durationEqualsMidnight* and *riseFromMidnight* constraints were added. The former constraint ensures that the starting time of a RiseTask will always be equal to midnight (00:00h). The duration of the RiseTask is the only variable that causes the RiseTask to have different ending times for each person. The ending time of the BedTime task always equals midnight (24:00h). Therefore the starting time seems flexible but in fact it is fixed. The duration of this tasks determines the starting time, which should always be equal to midnight minus the duration of the task. Both constraints are hard constraints. Schedules that violate this constraint are not considered to be valid.

Task type	Predecessor Tasks	Successor Tasks
Rise	None	All other tasks
BedTime	All other tasks	None
Shower	Rise	BedTime
Study	Rise	BedTime
Work	Rise	BedTime, Dinner
Dinner	Rise, Work	BedTime

Table 9: Predeccessor and successor tasks for each task type

4.3 Schedule tightness measure

For the purpose of experimentation, a measure was designed to capture the tightness of a schedule. This measure encapsulates different aspects of the schedule. Solely measuring the free time within a schedule is not sufficient to explain that schedule's flexibility. If all tasks in a schedule are placed consecutively, without any gaps in between, the schedule cannot be considered flexible. People are often not willing to alter their schedule in drastic ways. Therefore, it will be hard to move tasks around within this block of consecutive tasks.

The tightness measure consists of three submeasures that all describe a different inflexibility of the schedule. All measures are calculated for each person's personal schedule in the household. Hereafter, the submeasures are weighted and combined to one value for each person. The final value is the average of the values of all agents.

The first submeasure calculates the ratio between the occupied time and the total time within the personal schedule. A day always lasts 24 hours, therefore the total time in one's schedule is fixed. However, it is measured in minutes instead of hours. The occupied time is the sum of the durations of all tasks in minutes in one's schedule. The shower task time was not counted as occupied time, because we are looking at the flexibility of a schedule with respect to the showering tasks. The ratio is created by diving the occupied time by the total amount of time.

The second submeasure measures the free time in minutes within a personal schedule. Free time is considered to be the time of the day where no tasks have been scheduled. We will call this the empty space. However, since we are mostly interested in the flexibility of the shower tasks, we subtracted the time the shower was unavailable to the agent from the empty space. The shower is considered unavailable when another agent has scheduled a shower task during that time. The remaining free time in minutes is divided by the empty space. This ratio expresses the usefulness of an agent's free time with regard to showering. A higher ratio implies more useful free time and a more flexible schedule. However, the total measure measures the tightness of a schedule instead of the flexibility. Therefore, we subtract the ratio from 1.0 to correctly measure the tightness. This measure will be referred to as the shower ratio.

The third measure evaluates the number of consecutive tasks. Two tasks are considered to be consecutive, when one task's ending time equals the other task's starting time. The total number of consecutive tasks is divided by the total number of tasks minus 1. The latter value describes the number of possible connections between tasks. Dividing the two values results in a ratio that measures how flexible a schedule is when pushing tasks.

$$t(s) = w_1 * s_{occupiedRatio} + w_2 * s_{showerRatio} + w_3 * s_{consecutiveRatio} \quad (1)$$

The three submeasures were combined using weights which express the importance of the submeasures, see Formula 1. The submeasures are ratios between 0.0 and 1.0. The sum of the weights also equals 1.0 and therefore the resulting total is always a value between 0.0 and 1.0. The most flexible schedule has a tightness value of 0.0, whereas the tightest schedule has a value of 1.0. In our model, no basic schedule with an tightness value of 0.0 can be produced. Every generated schedule always assigns at least four tasks to each occupant: Rise, BedTime, Shower and Dinner task. All these tasks are also assigned a positive duration. Therefore, the occupied ratio measure can never be 0. Our model can potentially generate a schedule with a tightness measure of 1.0. However, this is very unlikely. For each person in the household, all three submeasures would have to equal 1.0. This can only occur when one's schedule is fully occupied.

The weights were chosen after some qualitative experimentation. The weights were tweaked until schedules with visually different tightness were assigned distinctive enough values. This distinctiveness was based on intuition and the design of the negotiation process. During negotiation, tasks can be reduced in duration and their starting time can be pushed. If an agent has a very busy schedule, meaning much of his/her time is occupied by tasks, it will be difficult to push tasks to make room for the event. For this reason, the occupied time ratio was seen as the most important submeasure. If an event requires to fit in a new shower, this event is constrained by the occupation of the bathroom. Therefore, the shower ratio was included. However, this submeasure is a very specific restriction and therefore less important than the occupied ratio submeasure. The number of consecutive tasks can complicate the process of pushing tasks. However tasks can still "jump" over other tasks. Therefore, this submeasure seemed to be the least important and was assigned the smallest weight. As a result, the weights in Formula 1

Person	Tightness
Person 0 (Adult)	0.67
Person 1 (Adult)	0.67
Person 2 (Child)	0.89
Person 3 (Child)	0.89
Person 4 (Child)	0.89
Average	0.80

Table 10: Tightness of household Figure 3c

were set to $w_1 = 0.7$, $w_2 = 0.2$ and $w_3 = 0.1$.

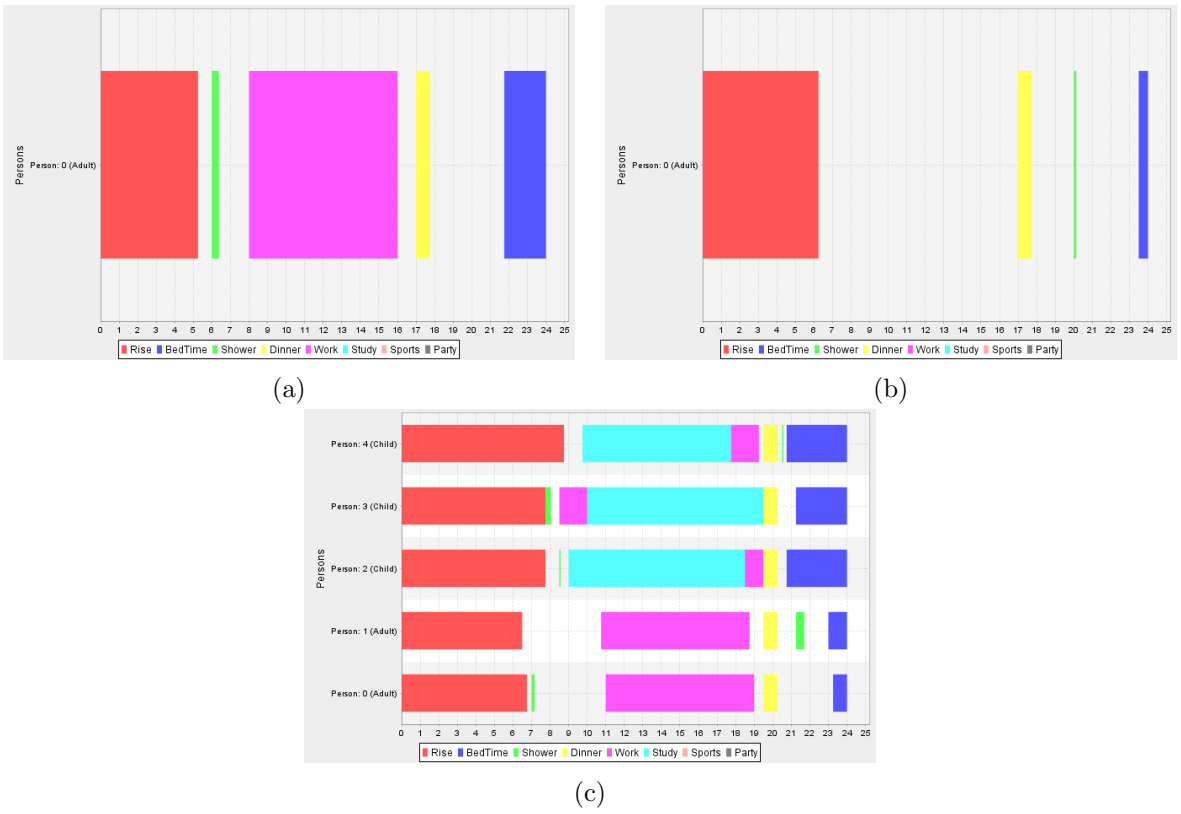


Figure 3: Basic schedules for three different households

In Figure 3, three schedules are shown, each for a different household. Figure 3a and 3b show schedules for single person homes. The person in 3a has a full time job, while the person in 3b is unemployed. Preferably, busier schedules such as the schedules of full-time workers, are assigned a higher tightness value. The designed tightness measure captures this effect with the occupied time submeasure. The found tightness values for 3a and 3b are 0.67 and 0.42 respectively. The measure successfully assigned a larger value to the tighter schedule.

Figure 3c shows a schedule for a 5 person home. The children in the household (person 2 - 4) all go to school and have a part-time job. This results in a visually tight schedule. The parents (person 0 and 1) seem to have more flexible schedules, especially in the morning. The calculated tightness values are shown in Table 10. As shown in Figure 3c the children are assigned a higher value, whereas the parents receive a lower value. This affects the average tightness of the schedule which is equal to 0.80. This is still a higher value than the single person home schedules.

5 Experimental setup

The developed model will be used to answer our research question by generating several thousand simulation runs and observing the different outcomes of the negotiation process. In this section we will show the simulation run setup and describe which results are stored and analyzed.

As explained in the Section 3 one simulation run will generate a household, create a basic schedule and negotiate some alterations to the schedule. A total of 4000 runs was performed with the parameter settings shown in Table 11. Each run all parameter values were chosen at random. Several parameters cannot be selected or randomly generated. The tightness of the basic schedule is calculated during simulation and the number of occupants in a household depends on the hType. As shown in the Table 11 not all household types were included. Several hTypes removed because the composition is the same as another hType. For example, hType 1 and 10 both consist of one full-time working adult. Separate runs do not influence each other and are therefore independent. Some simulation runs were not included in the final dataset because they violated one or multiple hard constraints. Runs that violated soft constraints were not removed.

Parameter	Value range
hType	[1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 15, 16, 19, 21, 22, 24]
Negotiating Agent	Random eligible occupant
eventType	[earlyMeeting, workOut, eveningParty]
Agent flexibility	[0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
Flexibility increase	[0.3]

Table 11: Value ranges of each simulation parameter

The experiment will consist of two parts: a quantitative analysis followed by a qualitative analysis. The quantitative analysis is based on several thousand samples generated by the developed model. During this phase the simulation run results of the scheduling and negotiation process were stored in a dataset. Table 12 shows a snapshot of a few rows of the stored results. Each simulation is assigned a unique ID. This way the matching pictures and other corresponding results can be found. The score column indicates the score of the basic schedule solved during the scheduling phase. Only the results of simulation runs that did not violate any hard constraints were saved. The total tightness of the basic schedule, as well as the tightness of the negotiating agent were stored in the table. Other results that were saved are the flexibility of the agent's during negotiation, the flexibility increase of the negotiating agent, the number of occupants living in the household, the household type, the event type generated during negotiating and the negotiator type. The flexibility increase was not varied during experimentation. It was assumed that the agent requesting alterations should be somewhat more flexible than his/her fellow housemates. The negotiator type encodes the type of person that requests the event. The code consists of three attributes: *adolescence_occupation_studentstatus*. Adolescence can be set to *adult* or *child*. The occupation options are FT for full-time workers, PT for part-time workers or N for unemployed people. The studentstatus is indicated by a boolean.

The final two columns are the most important column for the quantitative analysis. The $Successful$ column indicates whether negotiation was completed successfully during the simulation run. If an event could not be generated or an event could not be handled successfully the column is assigned the value No. Successful negotiation is indicated by the value Yes. The $RequiresNegotiation$ column indicates whether the requested event could be fit in the schedule without altering the schedules of other agents. These final two column will be used as the predictor variable during quantitative analysis. Samples will be divided in three classes: unsuccessful negotiation, successful negotiation that requires no alterations and successful negotiations that

requires at least on alteration.

ID	Score	Tightness	TightnessNegotiator	Flexibility	FlexIncrease	nrPersons	hType	eventType	negotiatorType	Successful	RequiresNegotiation
1551360553622_ID8	0hard/0soft	0.447917	0.447916667	0	0.3	1	9	eveningParty	adult_N_F	Yes	false
1551360553714_ID9	0hard/0soft	0.589375	0.739583333	0.2	0.3	5	24	eveningParty	adult_FT_F	Yes	true
1551360553841_ID10	0hard/0soft	0.611979	0.68125	1	0.3	2	5	earlyMeeting	adult_FT_F	No	false
1551360553953_ID11	0hard/0soft	0.6875	0.622916667	0.4	0.3	4	22	eveningParty	adult_FT_F	Yes	true
1551360554110_ID12	0hard/0soft	0.658333	0.779166667	0.2	0.3	4	22	eveningParty	child_N_T	Yes	true
1551360554270_ID13	0hard/0soft	0.725	0.717708333	1	0.3	5	2	workOut	adult_FT_F	Yes	true
1551360557093_ID14	0hard/0soft	0.673333	0.659375	0.2	0.3	3	19	workOut	adult_FT_F	Yes	true
...	...	...	...	...	...	...	...	...	...	...	...

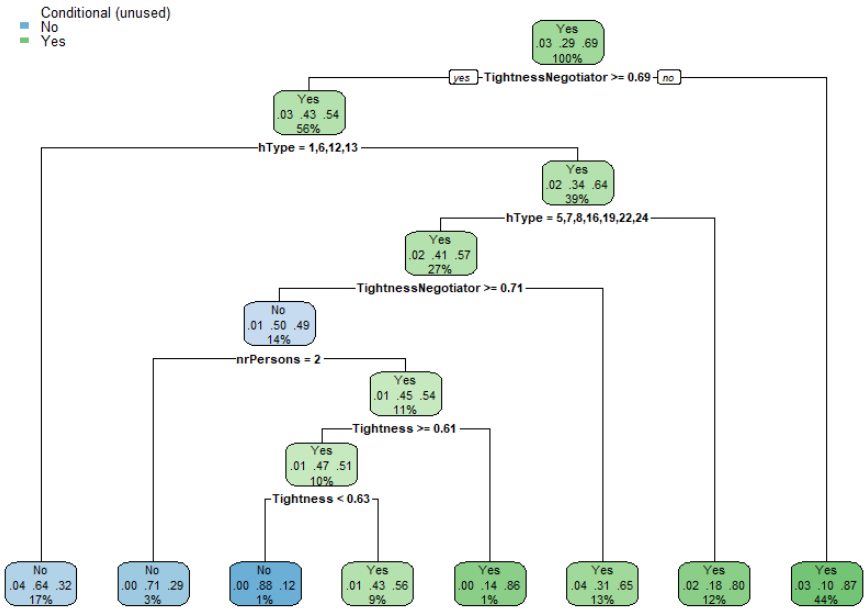
Table 12: Several rows of the dataset used for quantitative analysis

The simulation run dataset was processed and analyzed using classification models. Several models were tested but it was found that decisions trees formed the most interpretable model in this scenario. It also allows for dealing with interaction variables, as a tree structure allows for interactions automatically. Several single decision trees were build on the data. The dataset was first split in three subsets where each subset contains all data for a specific eventType. Each subset was split in a trainingset (70%) and testset (30%). A decision tree model was trained on each trainingset and evaluated on each testset. This will provide in an indication to whether the models are reliable representations of the data. This quantitative analysis is performed to find the most important variables that influence the negotiation process. The results obtained from this analysis will be used to select certain scenarios for qualitative analysis.

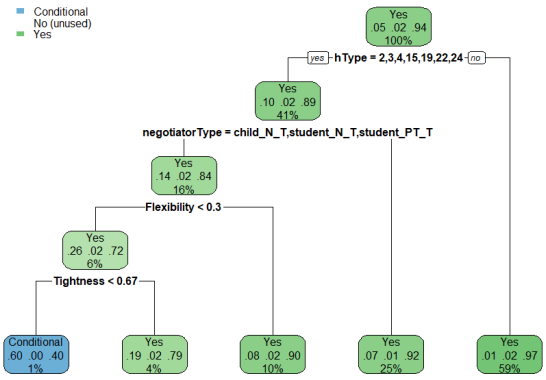
The decision trees models were built using the Gini index as the impurity function. No early stopping conditions were used, which allows the tree to grow maximally. Afterwards, the tree is pruned to prevent overfitting and to apply to more generic scenarios. A generic model is necessary to explain the factors that generally influence the negotiation process and not just in specific scenarios/households. However, we did split the data by event type because each event represents a different scenario in which different factors could influence the negotiation.

The earlyMeeting represents scenarios where an existing tasks needs to be extended and a shower should be scheduled in advance of this task. The eveningParty also represents the shower in advance scenario with the difference that an extra task has to be added to the schedule. The workOut event also requires this additional task but requires a shower afterwards. The factors in these events might be different. The events are also scheduled during different times of the day, which can again lead to different influencing factors.

6 Results



(a)



(b)

(c)

Figure 4: Decision trees for the earlyMeeting (a), eveningParty (b) and workOut (c) events

During the quantitative analysis it was found that the majority of the samples succeeded in scheduling an event without further changes to other tasks, they were labeled with *Yes*. The other two categories, labeled with *No* and *Conditional*, represent only 9.4% and 3.6% of the data respectively. Figure 4a, 4b and 4c show the decision trees for the earlyMeeting, eveningParty and workOut event for this data. Each tree has been pruned using the complexity parameter with the smallest cross validation error. This will create a more generic tree model. Pruning also allows us to analyze simply the most important splits. The variable importance of each variable in the datasets were also calculated, they are shown in Table 13 and 14. The importance of a variable is the sum of the decrease in impurity. This decrease in impurity is only counted when the variable was one of the most effective split variables in a node. The importance of all variables is converted into a percentage scoring. The most important variables for each eventType will be shown according to the variable importance measure and the resulting tree.

In figure 4a the decision tree for the earlyMeeting event data is shown. The model classifies

all samples as either *Yes* or *No*. No nodes in the tree have *Conditional* as its majority class as this class only represent a small proportion of the data. Both the tree and Table 13 show that the tightness of the negotiator and the hType are the most important variables. The top four splits are made by these variables. The tightnes values used in the splits are very similar values. If the tightness of the negotiator's schedule does not exceed 0.69, most cases do not require any negotiation. If they do exceed this threshold and are part of hType 1,6, 12 or 13 the event is most likely not successfully scheduled. These households do not have any school-going children or students as occupants, which the other hTypes do have. The next split on hType separates homes with several children or students and classifies them as no negotiation needed. The remaining households are split on tightness several times and shows that households with a greater tightness value and two occupants are more likely to fail negotiation.

In figure 4b the decision tree for the eveningParty event is shown. This model did not exclude the *Conditional* category however the *No* category no longer appears in the tree. The tree and Table 14 show that hType is the most important variable. The second variable in the table does not correspond to the second split in the tree. The tightness measure is responsible for separating the *Conditional* cases from the *Yes* cases which decreases the impurity of the tree and is therefore rated as a more important variable. In general, the importance percentages of all variables are very small which implies that these variables cannot differentiate among the three classes with high accuracy. This is most likely also due to the overrepresentation of the *Yes* class in the data. Similarly, the decision tree model built on the workOut event data cannot differentiate the different classes at all, as shown in Figure 4c. The fully grown tree had several more layers however each leaf nodes would have the *Yes* class as its majority class. This shows the effect of overrepresentation of one class in the data.

Another quantitative analysis was performed to gain more information regarding the negotiation process. The data was filtered to prevent overrepresentation. The samples that did not require any negotiation and could fit the event immediately were removed from the data. This leaves only the samples where negotiation was initiated somewhere during the event scheduling. The data was then split in two categories: successful negotiation and unsuccessful negotiation. It was found that in most cases the negotiation was completed successfully. Only 2% percent of the samples failed to schedule the event. All of these cases were workOut events. This covers 6% of the entire workOut event samples. A decision tree model was trained to classify successful versus failed negotiation. However, since hardly any cases failed the negotiation the dataset was to small to measure the influence of variables on negotiation success. The resulting decision tree consisted of a single node.

TightnessNegotiator	hType	Tightness	nrPersons	Flexibility
43.1797075	26.7363821	12.4931399	9.1114118	0.1012893

Table 13: Variable importance for earlyMeeting decision tree

hType	Tightness	nrPersons	negotiatorType	Flexibility	TightnessNegotiator
4.9641709	3.6815787	3.2257240	2.4426468	2.3350746	0.7499764

Table 14: Variable importance for eveningParty decision tree

A third comparison was made based on only the successful cases. The samples where the event scheduling was processed successfully can be split in two groups. The cases where negotiation was required to fit the event and the cases where the event could immediately fit without changes to other agents' schedule. For each group the means of the numerical features were calculated. These features are the schedule tightness, the tightness of the negotiator's personal

schedule and the flexibility of the agents. This test was performed to estimate the influence of the tightness and the flexibility parameters on negotiation. The means were calculated for each event type separately. Furthermore, the samples where an event could not be scheduled successfully were excluded from this analysis. This includes the samples where no negotiation was initiated because the requesting agent could not fit the event in his/her personal schedule and the samples where event negotiation failed because of the other occupants. The first case was not included because they should not be in the the same category as the schedules that have enough room to schedule the event immediately. Otherwise, the means might be distorted. For the second case, the data was shown to hardly contain any failed negotiation samples. Therefore, these samples will not influence the mean and can be removed without distorting the data.

For the earlyMeeting event, the means of the tightness, negotiator tightness and the agent flexibility per group were compared by conducting independent t-tests. No significant difference in tightness for the negotiation required group ($M = 0.678$, $SD = 0.065$) and no negotiation required group ($M = 0.691$, $SD = 0.057$) was found, conditions; $t(30) = -1.09$, $p = 0.283$. Similarly, there was no significant difference in negotiator tightness for the negotiation required group ($M = 0.680$, $SD = 0.0330$) and no negotiation required group ($M = 0.681$, $SD = 0.039$) conditions; $t(31) = -0.16$, $p = 0.872$. Furthermore, no significant difference for flexibility was found for the negotiation required group ($M = 0.441$, $SD = 0.331$) and the no negotiation required group ($M = 0.514$, $SD = 0.346$) conditions; $t(31) = -1.16$, $p = 0.255$.

The same independent t-tests were performed for the workOut event. No significant difference in tightness for the negotiation required group ($M = 0.684$, $SD = 0.064$) and no negotiation required group ($M = 0.667$, $SD = 0.077$) was found, conditions; $t(51) = 1.81$, $p = 0.075$. Similarly, there was no significant difference in negotiator tightness for the negotiation required group ($M = 0.658$, $SD = 0.0986$) and no negotiation required group ($M = 0.660$, $SD = 0.089$) conditions; $t(49) = -0.12$, $p = 0.902$. Furthermore, no significant difference for flexibility was found for the negotiation required group ($M = 0.528$, $SD = 0.327$) and the no negotiation required group ($M = 0.493$, $SD = 0.346$) conditions; $t(50) = 0.71$, $p = 0.480$.

The same independent t-tests were performed for the eveningParty event. A significant difference in tightness for the negotiation required group ($M = 0.695$, $SD = 0.061$) and no negotiation required group ($M = 0.668$, $SD = 0.075$) was found, conditions; $t(69) = 3.45$, $p = 0.0009$. Similarly, there was a significant difference in negotiator tightness for the negotiation required group ($M = 0.713$, $SD = 0.074$) and no negotiation required group ($M = 0.665$, $SD = 0.102$) conditions; $t(71) = 4.86$, $p = 6.822e - 06$. Furthermore, no significant difference for flexibility was found for the negotiation required group ($M = 0.449$, $SD = 0.320$) and the no negotiation required group ($M = 0.511$, $SD = 0.341$) conditions; $t(67) = -1.46$, $p = 0.149$.

The remaining features in the dataset are categorical values. Several chi squared tests were performed to find dependencies between a categorical variable and the necessity to negotiate. However, for many variables no accurate chi squared approximation could be calculated. This is due to the lack of examples for some of the combinations of variables.

A chi-square test of independence was performed to examine the relation between necessity of negotiation and number of occupants for eveningParty events. The relation between these variables was significant, $X^2(4, N = 1371) = 32.961, p < 1.216e - 06$. Households with three or more persons are more likely to require negotiation for eveningParty events than smaller households.

A chi-square test of independence was performed to examine the relation between necessity of negotiation and number of occupants for workOut events. The relation between these variables was significant, $X^2(4, N = 1297) = 49.208, p < 5.285e - 10$. Households with four or more persons are more likely to require negotiation for workOut events than smaller households.

Finally, some examples will be presented to show how certain circumstances can complicate the process of negotiation. The examples are shown in Figure 5. The left pictures show the basic

schedules before negotiation and the right pictures show the resulting situation after negotiation.

Figure 5a shows a 4 person household with hType 16. The type of people living in this household can be found in Appendix B. The score of the basic schedule is 0hard/0soft, the agent's flexibility is 0.2 and tightness and negotiator's tightness are 0.6029 and 0.6958 respectively. Person 2 requests a eveningParty event and wants to shower beforehand. The party starts at 21:00h and Person 0 would like to shower for 9 minutes starting at 20:51h. This event fits in his/her personal schedule, but conflicts with Person 1's showering practice. As Person 1 uses the shower for the purpose of relaxation his/her flexibility is reduced. This results in a flexibility of 0.0. Therefore, the agent is not willing to reduce the duration of the shower practice. The negotiator agent tries to relocate his/her shower task and suggests 20:42h with a duration of 9 minutes. The shower still fits in the agent's personal schedule, but still overlaps Person 1's shower task with 6 minutes. The negotiator tries to his/her show push 6 minutes to the left, starting at 20:36h. The shower fits and does not conflict with any other showering practices. The event is resolved successfully.

Example 2, as shown in Figure 5c and 5d, shows how a showering practice can fit in a very busy schedule. In this five person household of hType 2, Person 4 tries to schedule an eveningParty event. If a party task does not fit in an agent's schedule, an agent is allowed to reduce its sleeping task to fit the party. The party is scheduled at 21:00h with a duration of 90 minutes. The agent's dinner tasks ends at 20:45h and the requested shower has a duration of 6 minutes. This duration is quite short, but this is due to the meaning that is attached to the showering practice. An eveningParty event generates a showering practice with the freshness which generally results in short showers. Person 3's showering task is scheduled to start at 20:45h, right after dinner, with a duration of 8 minutes. Preferably, requesting agents schedule their shower right before the party task. Since the agent requested a shower of 6 minutes, this indicates a starting time at 20:54h. The shower fits in the basic schedule as Person 3 finished his/her shower 1 minute before.

The third example, depicted in Figure 5e and 5f, shows the schedules before and after a workOut event. The five person household (hType = 2) has a tightness of 0.764 and a negotiator tightness of 0.674. The agent's have maximum flexibility (=1.0) and the majority of the agents prefers to shower in the evening. Coincidentally, Person 0 is the only person who showers in the morning and is also the one with the smallest tightness value. Preferably the workOut event is scheduled after work but the gap until dinner is to small and is therefore scheduled after dinner. Person 0 tries to schedule a shower right after the sports practice but all other agent also shower in the evening. Person 0 sends a request to start his/her shower at 20:30h with a duration of 8 minutes. However, Person 3 has already scheduled a 4 minute shower during this time, but offers to reduce his shower by 2 minutes due to his/her high flexibility value. This offer is not sufficient, so Person 0 tries to push his shower to the right by 2 minutes, right after Person 3's shower. Even though every occupant showers around the same time, no new conflicts arise.

Figure 5g and 5h show a rare example where negotiation with more than one person was required. In this four person household (hType = 16), Person 0 tries to schedule a workOut event in the morning. Usually, the event is scheduled in the evening after work or study but since Person 0 is unemployed it is scheduled the morning instead. In the basic schedule, Person 0 and 2 usually shower in the evening while Person 1 and 3 shower in the morning. The total tightness and negotiator tightness are 0.56 and 0.45 respectively and the agent's have a flexibility value of 0.6. The negotiator agent requests a shower at 07:30h with a duration of 9 minutes. However, this causes a conflict with Person 3 who showers from 07:30h till 07:43h. Person 3 is willing to start 2 minutes later. This does not resolve the conflict for Person 0 and he/she tries to schedule a shower starting at 07:39h instead. This still overlaps with Person 3 shower, but it conflicts with Person 1's schedule as well. Person 1's shower starts at 07:45h with a duration of 21 minutes, but offers to shower from 07:48h instead. Person 3's old offer is removed and replaced by a new offer. Instead of starting 2 minutes later, Person 3 is willing to finish his/her

shower 3 minutes earlier. Person 3 was willing to reduce more than Person 1 even though the former's shower is shorter. This is due to the different meanings attached. Person 1 takes a shower for relaxation purposes while Person 3 showers to freshen up. The reduction is small but offers just enough space for Person 0 to fit his/her shower in between the other two showers.

Figure 5i and 5j show an example of a successful earlyMeeting negotiation. The three occupants all prefer to shower in the morning. The tightness of the schedule is 0.703 and the tightness of the negotiator's personal schedule is 0.623. Person 1's schedule is also the least tightest schedule due to less sleeping hours. The negotiator agent working time is extended and advanced by one hour from 09:00h to 08:00h. The agent requests a shower starting at 07:51h with a duration of 9 minutes. However, this overlaps with Person 1's schedule who showers from 07:45h till 07:53h. The agent is flexible (flexibility = 0.8, meaning = cleanliness) and offers to finish his/her shower 2 minutes earlier. This counteroffer is sufficient to fit Person 1's requested shower. Person 2's shower only starts at 08:00h which directly connects all three showers.

Figures 5k and 5l show an example of a schedule where negotiation failed. Person 0 planned a sports activity in the evening and tried to schedule a shower afterwards. The agent requested a shower with a duration of 11 minutes. This request fits in the agent's personal schedule as the gap between the SportsTask and BedTime task is 15 minutes long. However, Person 1 also scheduled a shower during this time. A simple solution could be to push Person 1's shower task. However in the current model other agents besides the negotiator are not willing to relocate their tasks due to implementation limits. But they are willing to reduce their showertask. Person 1 sends a counter offer to Person 0 offering to reduce its shower time by several minutes. However, this reduction is not sufficient to fit the agents requested shower. Person 0 tries to relocate the Sports task but fails due to his tight evening schedule. The agent's flexibility does not allow a sufficient reduction of its shower task either. Negotiation fails and the event remains unresolved.

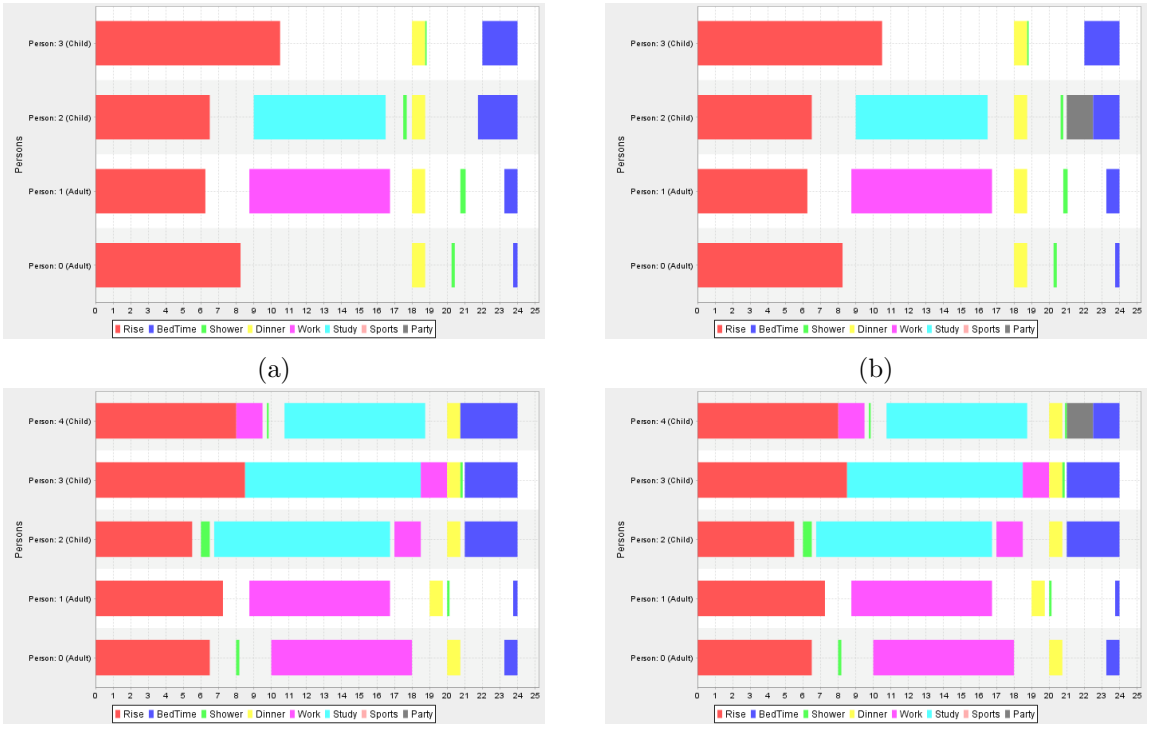
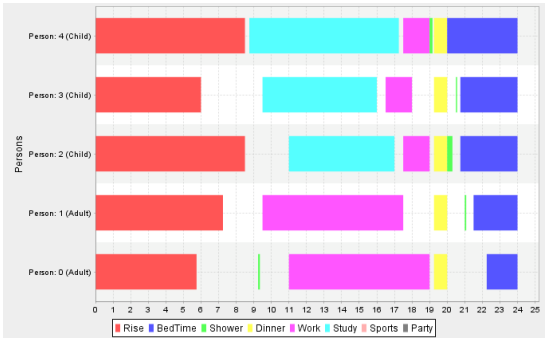
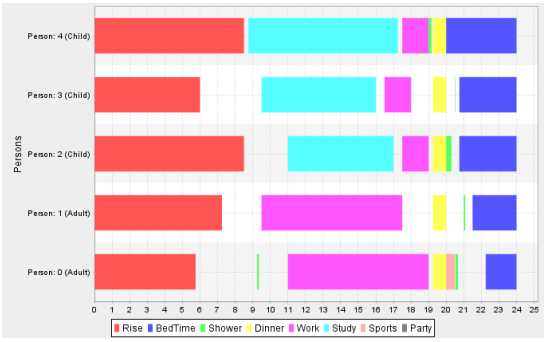


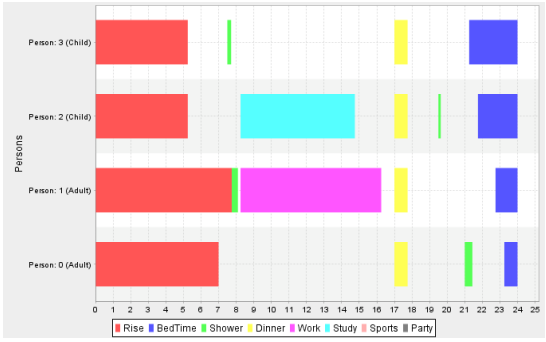
Figure 5: Examples of negotiation



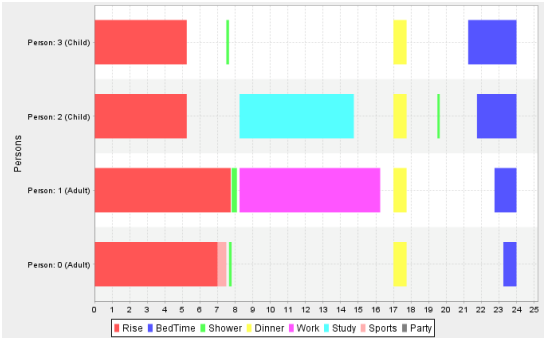
(e)



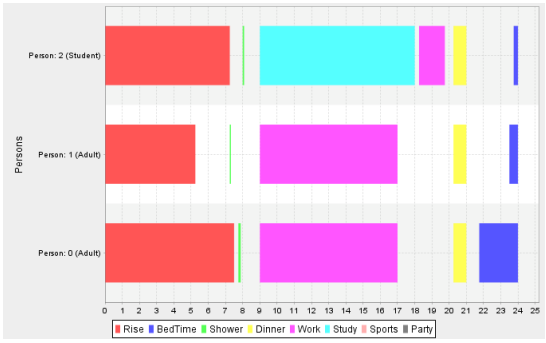
(f)



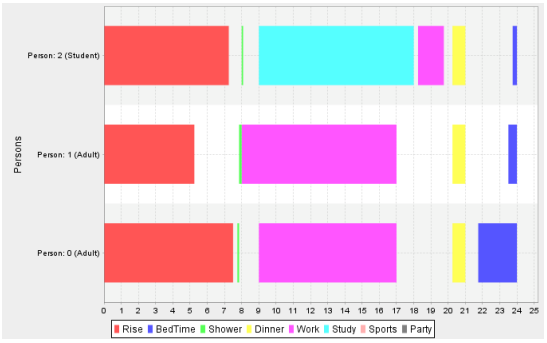
(g)



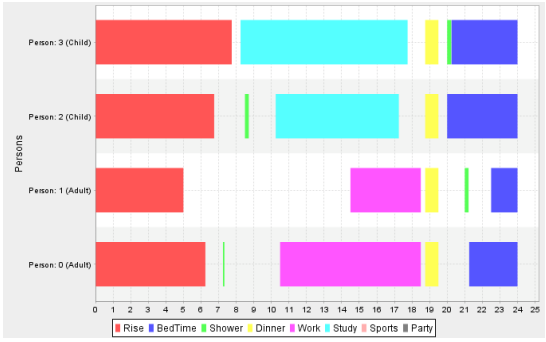
(h)



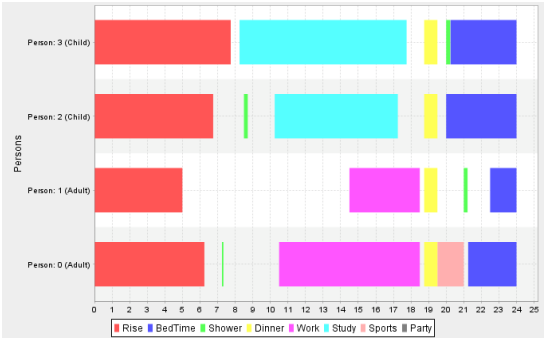
(i)



(j)



(k)



(l)

Figure 5: Examples of negotiation (cont.)

7 Discussion

In this section the implications of the results and shortcomings of the model will be discussed. It was shown that when the samples were split in three categories, Yes, No and $Conditional$, the majority of the samples belonged to the Yes class. The decision tree built on this data showed that it was difficult to distinguish between these classes due to this overrepresentation of the majority class in the data. In none of the trees the $Conditional$ samples were split from the No samples. If the differences between these to classes were to be measured, more data is required. However, simply generating more samples does not solve the problem because this would also generate many more Yes labeled samples. More No and $Conditional$ samples should be generated without generating a huge amount of Yes samples. The model should be extended to create tighter schedules that require negotiation with the other occupants more often. The analysis where the data was split between successful and failed negotiation also showed that more data is required as only 2% of the cases failed to schedule the event.

The earlyMeeting and eveningParty decision tree use a combination of tightness and flexibility thresholds to distinguish between samples that require negotiation and samples that can place the event without modification. This indicates a two-way interaction between the flexibility and tightness variables, which implies that when a schedule is tighter a higher agent flexibility is required to successfully adjust the schedule. Currently this only holds for very few cases of the earlyMeeting and eveningParty data. More data on negotiation samples is required to further investigate this two-way interaction. It should also be noted that the tightness values of schedules that allowed for instant modification of the schedules are very similar to the tightness values of schedules that failed negotiation. This is shown in the earlyMeeting decision tree where splitting on the negotiator tightness occurs twice with very similar values.

The third analysis where data was split between samples that required negotiation and samples that could instantly fit the event was used to test the influence of each variable on this split. None of the numerical variables showed a significant effect for the earlyMeeting or workOut data. For the eveningParty there was a significant difference in tightness and negotiator tightness for the required negotiation and instant fit class. The tightness and negotiator tigthness values were found to be slightly higher for the required negotiation class. This implies that if a schedule is busier, negotiation is more likely to be required to schedule an eveningParty event. This is contradictory to the other two events, where tightness showed no significant difference between classes. This difference might be caused by the time during which an event needs to be scheduled. The eveningParty event is set to be scheduled during the evening, while the earlyMeeting event is scheduled during the morning. The workOut event should be scheduled after the agent's final task before dinner, this could be either during the morning if the agent is unemployed or the evening of the agent is employed or is a student. Generally, the evening in the schedules seems busier as more tasks are scheduled during that time. The evening schedule is also heavily influenced by the other people in the household as the dinner task is preferably shared with the other occupants. Therefore, the dinner task is sometimes scheduled at later hours if one agent in the household works till late. This can cause the schedule of other agents to become tighter in the evening as the gap between dinner and bedtime decreases. If an agent then wishes to schedule an eveningParty task during this time the schedule is generally tighter during this time then other parts of the day. This local tightness could cause negotiation to be required more often.

The chi-squared tests showed that it was hard to measure the dependence between the categorical features and the necessity of negotiation. This was due to the lack of data for certain combinations of variables. Only for two of the variables the dependence relation could be shown. The requirement of negotiation for eveningParty events was shown to be dependent on the number of persons variable. A household with three or more persons is more likely to require negotiation to schedule an evening party than smaller households. Similarly, a dependence

relation was shown to exist by the workOut event negotiations and the number of persons variable. Larger households with four or more persons are more likely to require negotiation to schedule a sports task than smaller households. These dependency relationships show that the number of people in the household can complicate modifications to schedule. This implies that the number of people influences the interlockedness of the social practices pattern. For eveningParty it was already shown that tightness influences the need for negotiation. The number of persons could cause this higher tightness due to the shared dinner task. The tightness and number of persons variables are correlated. For the workOut event no significant difference in tightness was found. This might be due to the larger range of time during which the event can scheduled. The event can be scheduled in the morning where schedules are generally less tight. Or during the evening, where the scheduling can be influenced by the shared dinner task. Another difference between the eveningParty and workOut events is that agents do not have to wait for other agents to get home before they can start their sportstask. Party tasks are always scheduled after dinner, whereas a sports task can be scheduled before dinner if time allows it.

The qualitative analysis shows that meaning can complicate the negotiation process. If the schedule in Figure 5a was any tighter the requesting agent might not be able to push or reduce and the event scheduling would fail. Example 2 in Figure 5c showed a case where no communication among the agents was required. However, if one of the agents had scheduled or requested a shower with a different meaning or duration, negotiation would have been initiated and might have been a very complex process as the space is very narrow. In these cases, the flexibility of the agents come into play. But due to the low probability of these types of examples being generated, this complicated negotiation process hardly occurs. If cases like these were more likely to occur, the influence of the flexibility and the tightness parameter could be further tested. Creating tighter and more constraint schedules could increase the probability of such situations. Example 3 in Figure 5e shows another example of how people usually tend to shower around the same time in a household. However, it again shows that due to some really short showers and the negotiators loose schedule, resolving the conflict is easy. It also shows that negotiation is unlikely to happen among more than two persons as this example depicts a situation where the showers are scheduled in very close time range. Example 3 also shows that another occupant was willing to reduce his/her shower duration due to high flexibility. However, this reduction could have been avoided if the negotiator person had pushed his shower by 4 minutes instead of 2. This is caused by the orderly fashion in which the negotiation protocol is programmed. As an extension, the necessity of a duration reduction can be checked during the pushing procedure. Example 4 in Figure 5g shows that a high tightness value is not a requirement for multi-agent negotiation. It shows how tightness can be local which can cause a tough negotiation process. Example 5 in Figure 5i shows how tight morning schedules do not always require complex negotiation due to the short duration of shower tasks. A failed cases of negotiation was shown in example 6, Figure 5k. This is due to implementation limits and could be resolved in future extensions by allowing pushing and relocation of other agents' tasks.

In general the results imply that the tightness of a schedule does not influence the necessity or success of negotiation. However, this might be due to the lack of data on successful and failed negotiation. The dependency relationships of most categorical variables could also not be determined due to this lack of data. The influences of the numerical and categorical variables should be further tested by extending the model. More data on samples are required that involve negotiation. This could be achieved by creating tighter schedules. In the current implementation, generally schedules are too loose and therefore usually do not require negotiation or modifications to the basic schedule to fit an event. This could be achieved by adding more tasks to the schedules, such as house chores or by adding extra constraints to tasks such as meanings. Another addition could be parent-child relationships. For example, currently children do not need to ask their parents if they can go to a party. Another addition could be different types

of events, that take up more space in the schedule or require multiple participants, such as a birthday party for one of the occupants. These were not added to the current model given the timespan of this thesis.

Another improvement of the model addresses the tightness measure. A better tightness measure will have to incorporates other factors. No significant difference between the tightness of the schedules with and without negotiation was found. Whether a sample needs negotiation to fit an event should be reflected in someway by the tightness measure. It was shown that tightness seemed to influence the negotiation of eveningParty events but not the negotiation of the other event types. It was suggested that this could be due to the local tightness in the schedules. Therefore, the tightness measure could be improved by incorporating local tightness. Another factor to consider is the range of the tightness measure. Currently, the majority of the values fall in the range of 0.35-0.70. Therefore it is hard to differentiate between loose and tight schedules. Tightness values should be more distinctive for different classes.

Agent negotiation can also be extended with pushing and relocating of other agent's tasks. Furthermore, it could be extended by allowing reordering of tasks. Another improvement could be to check the pushing and reduction possibilities in a different order which can in some cases provide a solution where the current order cannot. However, these are edge cases as this event hardly occurred. If the schedules become more complex and are more interlocked these edge cases could be encountered more often.

Another extension of the negotiation model involves shower tasks. Shower tasks are shown to be very small tasks that have no trouble fitting in small time gaps in the schedule. The examples provided in Figure 5 show that when negotiation is required it is often easily solved with a very small adjustment. In reality, these small adjustments may not be made as people do not tend to clock their showers by the minute. More meanings could also be added to shower tasks. In the current implementation only cleanliness, freshness and relaxation are used but other constraints such as preference to shower before another person could be added. This could constrain the shower tasks a little further to lead to more interesting negotiations.

8 Conclusion

In Western Australia water shortage is caused by the Mediterranean climate. Many studies have been conducted to gain understanding in the behavior and technology that influence sustainable living. Many sustainable resource technologies have been developed and it was also tried to influence the behavior of individuals. However, this was soon found to be ineffective as human beings tend to fall back into old behavioral patterns after a while. It was suggested to shift the approach from individual behavior to the sustainability of everyday practices. Studies at CUSP are aiming to combine human behavior with the infrastructure of the home. The home is viewed as a interlocked system of social practices. Slightly adjusting one practice sometimes requires many changes in this pattern. Therefore, it is important to understand the connections among practices. Social practices are constrained by factors such as resources, meaning and skills of its carrier. These factors influence the flexibility of the social practice. It becomes more difficult to measure this flexibility once this practice is situated in an interlocked schedule of many other practices. Practices are interdependent as they can also share meanings, skills and resources with other practices.

In this thesis, we aimed to gain understanding in the flexibility of these social practices patterns. In particular, how tightness and occupant flexibility influence the flexibility of the social practices in a household setting. As the focus of this thesis lies on sustainability, the shower practice is the focal point. A social simulation was developed to approximate this flexibility of social practice patterns. The model was based on scheduling and agent-based modeling techniques. Scheduling was used to simulate the daily interlocked patterns practices within households. This allowed us to design several hard and soft constraints for the practices and the occupants. The resulting schedule was used as the basis for the negotiation protocol. This negotiation protocol allows agents to request changes to the original schedule. One agent could initiate negotiation and suggest a change. Other agents are able to respond to this change in the form of counteroffers. The success of the negotiation protocol is based on the constraints induced by the interlocked social practices.

An experiment was set up to measure the influence of several parameters on the negotiation protocol. A measure was designed to calculate the tightness of a schedule before negotiation. This measure incorporated the occupied time, the amount of time available for showering and the number of consecutive tasks. This tightness was measured for the total schedule and all agents individually. The simulation was run several thousands times. Numerical and categorical values were stored in a dataset for quantitative analysis. The basic and negotiated schedules were saved as images and as text for qualitative analysis. Several parameters such as the flexibility of agents were varied for each simulation run.

In general no significant relation was found for the tightness of the schedules and flexibility of the agents on the negotiation protocol. The dataset was deemed too small to measure influence of parameters on success of negotiation as majority of the samples succeeded to schedule the event without adjusting the other agents' schedules. The dependency relationships of most categorical variables could not be measured due to this lack of data. For earlyMeeting and workOut events tighter schedules and less flexible occupants did not increase the need for negotiation. However, for eveningParty events a significant difference in tightness and individual tightness for the requirement of negotiation was found. This implied that busier daily schedules are more likely to require negotiation to schedule party tasks. This holds for both individual tightness as well as tight household schedules. The absence of this significance relation for the other event types could be explained by local tightness. The eveningParty event requests modifications of a busier part of the schedule in comparison to the other event types. Another observation showed that households with more than three people for eveningParty events and households with more than four people for workOut events are more likely to require negotiation. This shows that the number of people influences the interlockedness of social practices and

complicates modifications to this pattern.

The qualitative analysis showed that the meanings attached to shower tasks can influence the negotiation process. The relaxation meaning was shown to complicate the negotiation because this meaning does not allow the duration of a shower task to be greatly reduced. The analysis also showed that most of the samples required no negotiation because the shower tasks are very small and are not very likely to overlap. Showers could also be scheduled in smaller gaps in between tasks as most showers only last 10-15 minutes depending on the meaning. In almost all cases where negotiation was required, the requesting agent would have to negotiate with only a single other occupant. Practically no multi-agent communication occurred. The multi-agent negotiation case showed that these cases are very unlikely to occur even if all agents shower during the same time period because showers are very short tasks and therefore are very unlikely to overlap. It also showed why flexibility was not found to be significant. Negotiation was not necessary to modify the schedule in most samples. Therefore, only in some cases the flexibility parameter would be useful. But even in the negotiation cases they could often be solved by simple 2 or 3 minute reduction of one of the shower tasks. One example was shown where negotiation failed. However, this failed case could easily be avoided by extending the model.

In general tightness and flexibility were found to not significantly influence the negotiation protocols. Therefore, tightness and flexibility do not seem to be the major constraints that interlock the pattern of social practices. This implies that the flexibility of human beings and the tightness of their daily lives are not the factors that restrain humans to change their behavior towards more sustainable living. However, the eveningParty showed that tightness and flexibility can be important factors for busier times during the day. This implies that in general the simulation schedules might not be interlocked enough to measure this influence for all event types. This could be studied further by extending the model.

The shower tasks could easily be modified to allow changes to the schedule. These practice are not restrained by any other practices and additional showers could easily be scheduled thanks to their short durations. The flexibility of the shower practices was slightly influenced by its attached meaning, such as the relaxation meaning. However, the presence of this meaning did not result in any major conflicts as it still allowed for modification.

In future research the model can be improved in three ways. The scheduler can be extended to increase the tightness of the daily schedules. This could be achieved by adding more basic tasks such as household chores. More constraints should be added to showers as they currently fit in the schedule very easily due to their short duration. Another extension is an improvement of the tightness measure. The tightness value of loose and tight schedules are very similar in its current state. The measure should also incorporate local tightness. Another item subject to future research is the negotiation process. It should be extended with more options to relocate and reduce tasks in other occupants schedules. An additional constraint could be extra shower meanings that allow relocation but not reduction or vice versa. All these extensions will allow for further study and improvement in the representation of interlocked social practice patterns.

References

Athanasiadis, I. N. and Mitkas, P. A. (2005). Social influence and water conservation: An agent-based approach. *Computing in Science & Engineering*, 7(1):65.

Dignum, F., Dignum, V., Prada, R., and Jonker, C. M. (2015). A conceptual architecture for social deliberation in multi-agent organizations. *Multiagent and Grid Systems*, 11(3):147–166.

Dignum, V. and Dignum, F. (2014). Contextualized planning using social practices. In *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pages 36–52. Springer.

Eon, C., Breadsell, J. K., Morrison, G. M., and Byrne, J. (2018a). The home as a system of practice and its implications for energy and water metabolism. *Sustainable Production and Consumption*, 13:48–59.

Eon, C., Liu, X., Morrison, G. M., and Byrne, J. (2018b). Influencing energy and water use within a home system of practice. *Energy and Buildings*, 158:848–860.

Eon, C., Morrison, G. M., and Byrne, J. (2018c). The influence of design and everyday practices on individual heating and cooling behaviour in residential homes. *Energy Efficiency*, 11(2):273–293.

Gilbert, N. (2008). *Agent-based models*. Number 153. Sage.

Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., Goss-Custard, J., Grand, T., Heinz, S. K., Huse, G., et al. (2006). A standard protocol for describing individual-based and agent-based models. *Ecological modelling*, 198(1-2):115–126.

Grimm, V., Berger, U., DeAngelis, D. L., Polhill, J. G., Giske, J., and Railsback, S. F. (2010). The odd protocol: a review and first update. *Ecological modelling*, 221(23):2760–2768.

Holtz, G. (2014). Generating social practices. *Journal of Artificial Societies and Social Simulation*, 17(1):17.

Kahneman, D. and Egan, P. (2011). *Thinking, fast and slow*, volume 1. Farrar, Straus and Giroux New York.

Lopes, M., Antunes, C., and Martins, N. (2012). Energy behaviours as promoters of energy efficiency: A 21st century review. *Renewable and Sustainable Energy Reviews*, 16(6):4095–4104.

Luo, L., Zhou, S., Cai, W., Low, M. Y. H., Tian, F., Wang, Y., Xiao, X., and Chen, D. (2008). Agent-based human behavior modeling for crowd simulation. *Computer Animation and Virtual Worlds*, 19(3-4):271–281.

Macrorie, R., Foulds, C., and Hargreaves, T. (2015). Governing and governed by practices: Exploring interventions in low-carbon housing policy and practice.

Marsella, S. C., Pynadath, D. V., and Read, S. J. (2004). Psychsim: Agent-based modeling of social interactions and influence. In *Proceedings of the international conference on cognitive modeling*, volume 36, pages 243–248.

Mercuur, R. (2015). Interventions on contextualized decision making: an agent-based simulation study. Master's thesis.

Mester, Y., Kökciyan, N., and Yolum, P. (2015). Negotiating privacy constraints in online social networks. In *Advances in Social Computing and Multiagent Systems*, pages 112–129. Springer.

O'Brien, L. V., Kashima, Y., and Walshe, J. (2017). Year 3 project report: Transformation to low carbon living - social psychology of low carbon behavioral practice (rp3012).

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer.

Reckwitz, A. (2002). Toward a theory of social practices: A development in culturalist theorizing. *European journal of social theory*, 5(2):243–263.

Schelling, T. C. (2006). *Micromotives and macrobehavior*. WW Norton & Company.

Shove, E. (2010). Beyond the abc: climate change policy and theories of social change. *Environment and planning A*, 42(6):1273–1285.

Shove, E., Pantzar, M., and Watson, M. (2012). *The dynamics of social practice: Everyday life and how it changes*. Sage.

Spurling, N. J., McMeekin, A., Southerton, D., Shove, E. A., and Welch, D. (2013). Interventions in practice: Reframing policy approaches to consumer behaviour.

Stimpson, J. L., Goodrich, M. A., and Walters, L. C. (2001). Satisficing and learning cooperation in the prisoner s dilemma. In *IJCAI*, volume 1, pages 535–540.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Young, H. P. (2004). *Strategic learning and its limits*. OUP Oxford.

Appendix A Class annotations

The classes in the Java project were annotated as instructed by the Optaplanner library.

Annotation	Class	Description of annotation
Planning Solution	HouseholdSchedule class	Objects of this type will contain all building blocks for the schedule. The solver will store the found solution in objects of this class.
Planning Entity Collection Property	TaskAssignment list in Household class	List of building blocks that need to be assigned by the solver.
Planning Fact Collection Property	TimeSlot list in Household class	List of building blocks that can be assigned by the solver.
Planning Entity	TaskAssignment class	Objects of this type are assigned a planning variable by the solver.
Planning Variable	Starting TimeSlot object in TaskAssignment class	This variable is assigned a value by the solver during scheduling.
Planning Score	HardSoft score object in Household class	In this object the solver will store the score of the found solution.

Table 15: Annotated classes in Java project

Appendix B Household type (hType) compositions

Codes used in Table 16:

- FT = full-time worker
- PT = part-time worker
- UE = unemployed
- home-Child = home-staying child, if not indicated school-going child

hType	nrPeople	Occupants
1	1	1x FT-Adult
2	5	2x FT-Adult, 3x PT-Child
3	3	1x FT-Adult, 2x PT-Child
4	4	2x FT-Adult, 2x Child
5	2	1x FT-Adult, 1x PT-Adult
6	2	2x FT-Adult
7	3	1x FT-Adult, 1x PT-Adult, 1x Student
8	2	1x FT-Adult, 1x Child
9	1	1x UE-Adult
12	4	1x FT-Adult, 2x PT-Adult, 1x home-Child
13	3	3x FT-Adult
15	4	4x Students
16	4	1x FT-Adult, 1x UE-Adult, 1x Child, 1x home-Child
19	3	2x FT-Adult, 1x PT-Student
21	3	2x UE-Adult, 1x Student
22	4	1x FT-Adult, 1x PT-Adult, 2x Child
24	5	1x FT-Adult, 1x UE-Adult, 1x Child, 2x home-Child

Table 16: Types of occupants living in each hType