Auton Agent Multi-Agent Syst (2016) 30:291–330
DOI 10.1007/s10458-015-9288-2

# Preference-based reasoning in BDI agent systems

**Simeon Visser · John Thangarajah · James Harland · Frank Dignum**

Published online: 18 February 2015


**Abstract** An important feature of BDI agent systems is number of different ways in which an agent can achieve its goals. The choice of means to achieve the goal in made by the system at run time, depending on contextual information that is not available in advance. In this article, we explore ways that the user of an agent system can specify preferences which can be incorporated into the BDI execution process and used to guide the choices made. For example, a user of a travel system can specify a preferred airline, or a particular kind of accommodation, and the system will use this information to satisfy the goal and preferences, if possible. Preferences are specified in terms of properties of goals and resource usage, and are used to make two types of decisions: (a) select a plan when there is a choice and (b) determine the order in which subgoals of a plan should be pursued when their order is not fixed by design. We have implemented our preference framework in Jadex, and provide detailed case studies within the context of a holiday travel agent application.

**Keywords** Agent programming languages · Reasoning (single and multiagent) · Preference reasoning

## 1 Introduction

The *Belief-Desire-Intention* (BDI) [22] model of agency is a popular agent development paradigm for developing complex applications. The agents are built using cognitive concepts such as goals, plans and beliefs. A typical BDI agent is equipped with a pre-defined set of *plans* which are recipes for achieving its *goals*. An agent generally has several plans that could be utilized to achieve a goal, and the choice is often dependant on the agent's *beliefs*. Plans generally contain subgoals, which are in turn handled by other plans. Hence, there are

S. Visser · F. Dignum
Utrecht University, Utrecht, The Netherlands

J. Thangarajah (✉) · J. Harland
RMIT University, Melbourne, Australia
e-mail: john.thangarajah@rmit.edu.au

generally several choices of plans to be made when achieving a particular goal. This means that the execution of a BDI agent can be thought of as a sequence of choice points, where each choice point involves choosing the next plan to be executed.

A key feature of these 'intelligent' agents, is the way in which the agent deliberates over its choices and how it manages the consequences of these decisions. For example, a travel agent that is asked to book a holiday may subdivide this task into two subgoals of booking accommodation and booking transport. If there are 5 different accommodation venues (possibly involving hotels, bed-and-breakfast and backpacker venues) and 6 different means of transport (such as two different trains and four different airlines), there are a total of 30 different combinations that may be used by the agent to achieve the goal. However, not every such combination may be available; for instance, a given flight may arrive too late for bed-and-breakfast accommodation to be possible, or a given hotel or flight may be full. It is precisely this uncertainty that motivates the use of the BDI approach, in that we can specify what needs to be done (booking accommodation and transport) whilst leaving the agent free to find the most appropriate combination of plans which will achieve this.

There has been work towards enabling BDI agents to make more rational choices, by introducing mechanisms to detect and avoid conflicts [29], identify and take advantage of synergies [27] and more recently on exploiting coverage and overlap measures for plan selection [28]. In this article we further this research and explore the use of *preferences* in the agent deliberation process.

In practice, it is common for the user to want to specify some preferences for how the goal should be achieved. For instance, in the travel example above, the user may wish to specify a particular choice of airline (to take advantage of a frequent flyer membership), or that it is preferable to travel by train and spend any money saved on a better class of accommodation. Note that this extra information is included as a preference rather than a goal since, it is acceptable to satisfy the goal without satisfying the preference. For example, if the user prefers to fly on Dodgy Airlines, but no such flights are available, then specifying this as a preference means that the user can still have a holiday; specifying this as a goal would mean that the user refuses to travel by any means other than Dodgy Airlines.

In this paper we present a way of allowing the user to specify his or her preferences over the execution of particular goals independent of the plans (and subgoals). The preferences are therefore not pre-programmed into the system and are provided by the user prior to execution. We provide detailed mechanisms for the specification of user preferences and incorporating them into the agent deliberation process. *The key benefit of our approach is that the same system can be used by different users with different preferences.*

We achieve this by adapting the preference language $\mathcal{LPP}$ [1,2] to specify preferences over *properties* of goals, which are a way of specifying what will occur when the goal is achieved. For example, booking a 5* hotel gives the property accommodation *class* the value 5*. The properties of a goal and their values are presented to the user, who can then use these to specify preferences, without having to know how the goals may be achieved.

In our approach preferences are used for two different types of deliberation: (a) plan selection and (b) subgoal ordering. Preferences can be used to choose plans that would satisfy the user's preferences over others. For example, if the user prefers 5* accommodation, then the agent should first attempt to book accommodation of this type, i.e. choose plans which book 5* hotels in preference to other plans. Similarly, when no ordering between subgoals is specified, we provide mechanisms which can exploit this to enable greater preference satisfaction. For example, to satisfy the preference of travelling by train and spending any money saved on accommodation, it is necessary that the subgoal of booking the train be performed first.

Often the choices made early in the achievement of goals affect later choices. Hence it is important to perform some form of look ahead analysis. Given that the future plan choices are not pre-determined we build on the the notion of *summary information* [27,29] to derive information at abstract levels that can be used at early stages of decision making.

This article is organized as follows. In Sect. 2 we discuss the use of preferences in agent platforms and other fields within computer science, such as automated planning. In Sect. 3 we discuss how the user preferences are specified in our preference language and what information is added to the agent specification to enable the user to express preferences. In Sect. 4 we discuss our reasoning algorithms which are built into the reasoning component of an agent to utilize the user preferences. In Sect. 5 we discuss our implementation of these techniques in an existing agent platform, Jadex [21], and demonstrate our work with case studies. Lastly, in Sect. 6 we discuss some of the limitations of our work and present directions for future work.

## 2 Background and related work

Preferences can be regarded as soft constraints on the solution of a problem or the performance of a system. These constraints are soft in the sense that they do not need to be satisfied although solutions in which they are satisfied are preferred. Extensive research has been conducted to utilize preferences in areas such as robotics and operations research. Preferences are contrasted with hard constraints that must be satisfied in a solution or performance of the system. The benefit of preferences in these areas of research is to improve the quality of the computed solutions to the problem.

### 2.1 Preferences in related areas of research

In operations research, a *constraint satisfaction problem* looks at formulating solutions to a problem as a set of hard and soft constraints. Each solution of the problem must satisfy the hard constraints while the soft constraints should be satisfied as well as possible. For example, a problem could be to pack some packages into a truck. In this problem a hard constraint could be to pack fragile packages in safe positions and a soft constraint (i.e., preference) could be to pack all packages in the smallest possible space. This preference aids in searching for a solution that packs the packages most tightly in the truck which leaves more space for any future packages that need to be transported.

In the area of *automated planning*, a sequence of actions, called a plan, needs to be found to reach a particular situation from an initial situation. The specification of the available actions with their effects, the initial situation and the desired situation is called a planning problem and any plan that reaches the desired situation is a solution to this problem. Preferences can be added to the problem specification to indicate that certain actions should be avoided or that certain situations should not occur while trying to reach the desired situation. For example, a cleaning robot should not move around needlessly or clean a part of the room that is already clean. There can be many sequences of actions that result in a clean room but striving to satisfy these preferences results in much more efficient plans being used.

Utilizing preferences in automated planning focuses on (1) expressing these preferences in the specification of the planning problem and (2) computing plans that satisfy these preferences as good as possible according to some metric. Some examples of languages for specifying planning problems are the well-known STRIPS language [8], the Planning Domain Definition Language 3 (PDDL3) [9,12,13] and the language $\mathcal{LPP}$ [1,2], upon which the

preference language in this work is based. The latter two of these languages support the specification of preferences.

In these areas of research the basic approach is to compute one or more solutions to the problem at hand (i.e., that satisfy the hard constraints or desired characteristics). Furthermore, the solutions are then compared according to how well they satisfy the soft constraints or preferences. For this a metric function is used which computes a numeric value based on how well the preferences are satisfied. A simple metric function counts the number of preferences that are satisfied. A high value for this metric function generally implies a good solution while a low value generally implies a solution that does not satisfy the preferences very well. In practice, more elaborate metric functions are used to make some preferences more important than others.

The task of automated planning shares some similarity to intelligent agent systems in the sense that both systems aim to bring about a certain outcome by taking actions. An important difference between these systems is that in automated planning the course of action is computed before any actions are taken whereas the actions of an intelligent agent are computed during execution.

We do not follow the approach of computing possible executions beforehand and then comparing them according to how well they satisfy the preferences. This would not be feasible as the number of possible executions grows quickly [31] and the future plan choices are unknown, which is the basic philosophy that underlies BDI agent systems. The ability to adapt is one of the key benefits of an intelligent agent and the reason why we wish to use preferences during an agent's execution rather than before.

### 2.2 Preferences in intelligent agents

In the context of intelligent agents, preferences have been used for, e.g., controlling the selection of plans and logical reasoning about actions to take. Preferences have also been used for the selection [14,19,20] and elimination [15,17,18] of actions or plans.

Nguyen and Wobcke [19] use preferences as a means for choosing between multiple applicable plans in the context of a dialogue system. When there are plan choices, the user is prompted to specify its preference by presenting a decision list with possibly nested if-then statements. Over time, the agent learns from these choices and adapts to the user's preferred choices. The preference representation is therefore, simply that the user prefers one plan choice over the other and the preference reasoning is local between a set of plans. Our approach is more sophisticated, in that we use a preference language that is able to capture preferences over properties, rather than individual plans. The preference reasoning performs a look ahead, taking into consideration all possible paths that may unfold if a particular plan is chosen, and deliberates on how that path satisfies the user's preferences. We do not learn plan choices but use predefined user preferences to reason over plan choices and subgoal ordering at run-time. This also means that our preferences may be used in more than one context.

Myers and Morley [17,18] use preferences to restrict the autonomy of agents to allow a human supervisor to direct and guide the tasks performed by a group of agents. Preferences consist of a context, and a constraint on the required and prohibited features of a plan. Preferences are used during the execution of the agent to eliminate some instances of applicable plans that an agent can select in response to a goal. Their work is implemented in a system called TRAC, which is built on top of the Procedural Reasoning System (PRS) [16]. The work of Myers and Morley focuses on three types of decision: the decision to respond to a new goal or event, the decision to select an applicable plan for a goal and the decision

of how to instantiate the variables of a plan. In our work, we focus on the second of these decisions. The decision of whether and how to respond to a goal or event does not occur in our system. The third type of decision, instantiating variables of a plan, is currently not part of the presented work although we believe the work can be straightforwardly extended if needed.

Our work differs from the work of Myers and Morley in that, similar to the work of Nguyen and Wobcke, their preference specification indicates the context in which an agent should select a particular activity specification (choice of plan and variable instantiations for designated roles). The limitation of their work is that the preferences are tightly coupled to the local plan choice similar to that of Nguyen and Wobcke, that is, specifying explicitly that $\text{plan}_x$ is preferred over $\text{plan}_y$. In our approach preferences are specified independent of the plans.

Toranzo et al. [30] provide a comprehensive mechanism for decision-making within a BDI agent based on argumentation techniques. Such choices include choosing between conflicting desires, reconsidering intentions, and choosing between plans. We only consider the last of these three, and Toranzo et al. do not consider re-ordering goals within plans. In addition, their techniques require a total ordering on alternative plans, which we do not, and there is no language specified for user preferences in their work. The total ordering is not possible in BDI agent systems where the future plan choices depends on the state of the world at that point in time.

Dasgupta and Ghose [7] provide a mechanism to represent preferences into the BDI agent framework by allowing the user to specify a preference value for each plan. Whilst this is one way of representing preferences and is adequate in some domains, it does not capture global preferences, and does not allow for specifying conditional preferences.

In contrast to the above work, in our approach ahead, we capture the global preferences of a user in terms of properties related to a domain (such as cost, airline quality, accommodation type etc. for a travel domain) and aim to select plans that satisfy the user's global preferences with respect to these properties.

Hindriks and van Riemsdijk [15] have developed a layered approach for integrating goals and preferences into the GOAL agent programming language.[1] Their work is based on linear temporal logic and provides a formalism for choosing among the available actions. Goals and preferences are respectively regarded as hard and soft constraints. Both types of constraints are modelled as a process of elimination of some of the available actions, in that goals eliminate actions that violate the goal condition and, similar to previously discussed approaches, preferences are used to eliminate some of the available actions. When the actions that the agent can select does not include any that satisfy the preferences, the preferences are ignored.

The aims of their work is similar to ours—when there is choice in the next course of action, select the one that maximises the user's preferences. The execution of a GOAL program is such that it is composed of a set of action-rules and performs an action at every step by evaluating the current state of the world. In contrast, BDI languages such as JadeX [21], JACK [4] and GORITE [23], have predefined plan libraries from which plans are chosen and executed to achieve particular goals. This means that they require a different reasoning mechanisms, such as the mechanisms based on summary information as proposed by Thangarajah et al. [29], which we adapt in our approach to reason about preferences.

Furthermore, they represent goals and preferences using operators of linear temporal logic while our approach utilizes a preference language that aims to be more suitable for end-users to express their preferences.

[1] http://mmi.tudelft.nl/trac/goal.

Hindriks et al. [14] augment the specification of an agent program with utility values that provide heuristic guidance in the agent's selection of actions. Padgham and Singh [20] provide a mechanism for providing preference values for each plan which are evaluated dynamically at run time. These approaches are appropriate for when the developer has the appropriate domain knowledge to encode the preferences. It does not however capture the general preferences of the users of the system. *Our aim is to express the user's preferences in a language that uses terms and concepts of the problem domain* (see Sect. 3), rather than a language which incorporates specific details of the agent.

The work of Fritz and McIlraith [10,11] shares some similarities with the work presented here. They use a subset of the preference language of Bienvenu et al. [2] (as we do). In our case, we integrate the use of preferences into the agent platform Jadex while they provide an integration of preferences into the agent programming language DT-Golog [3]. The practicality of their work is shown by means of a travel planner that utilizes user preferences and the Yahoo!-Travel online service. The main difference between their work and ours is that they are concerned with generating plans that conform to user preferences, whilst we are concerned about making the choice between plans at run time.

Our use of summary information for preference-based reasoning in BDI agents is the key novelty compared to all other previous approaches.

### 2.3 BDI agents and goal-plan tree structures

As mentioned in the Sect. 1, the focus of this work is on BDI agent systems. A typical BDI agent system consists of a plan library where for a given goal, there is one or more plans in the plan library that could possibly achieve the goal (this is an OR decomposition, as any of the plans can satisfy the goal). Each plan performs some actions and/or posts a number of subgoals (this is an AND decomposition as all subgoals must be achieved for the plan to succeed). A subgoal is in turn handled by some other plan in the plan library in a similar manner. This decomposition leads to a natural hierarchy which is termed a *goal-plan tree*. The root of a goal-plan tree is a goal node, and the leaves of goal-plan tree are plan-nodes. Each goal-node has one or more plan nodes as children, and each plan-node has zero or more goal-nodes as children. A goal is considered achieved if one of its child plans succeeds. A plan is considered achieved if every one of its child goals succeeds.

Figure 1 shows an example goal-plan tree of a goal of booking a holiday (HOLIDAYGOAL), which consists of two subgoals for booking accommodation (ACCOMMODATIONGOAL) and booking transport to the destination (TRANSPORTGOAL).

Goal-plan trees can be annotated with additional information, such as the pre- and post-conditions of the goals and plans, the resource consumption of the plans, etc. Previous work by Thangarajah et al. [27,29], and Clement and Durfee [5,6] have shown how to use annotated goal-plan trees to improve agent decision-making by avoiding potential conflicts and taking advantage of synergies. The key idea is that information at the lower levels of the tree can be summarised and propagated to the higher levels of the tree. This gives a form of look-ahead for BDI-agents. The propagated information is termed *summary information*.

Goal-plan trees have been analyzed by Shaw et al. [24–26] and they present various theoretical properties and experimental results on the performance of these techniques. Shaw et al. present a method for representing the goals and plans of an agent using mathematical models called Petri nets. In this approach they aim to perform similar reasoning about goals without utilizing the notion of summary information as developed by Thangarajah et al. and Clement et al.

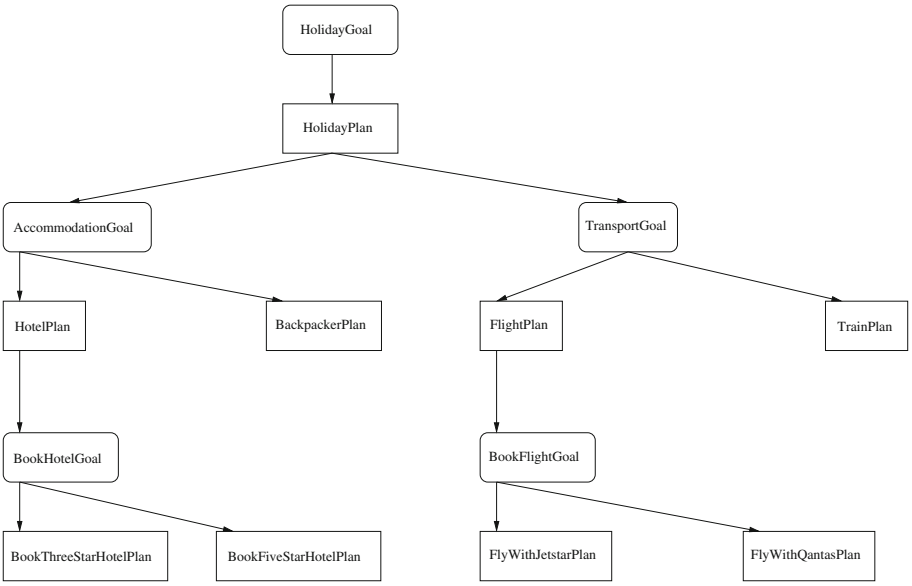


**Fig. 1** Goal-plan tree example

Furthermore, the characteristics and possible executions of a goal-plan tree have been analyzed by Winikoff and Cranefield [31], showing that the number of possible agent executions grows significantly as the goal-plan tree becomes bigger. Mechanisms such as plan failure profoundly contribute to the number of ways an agent could possibly be executed. A thorough theoretical analysis of preference satisfaction is beyond the scope of this work. However, the vast number of executions is something to keep in mind when providing a formal analysis of our preference system, as mentioned in our discussion of future work in Sect. 6.

In this work, we build on the summary information approach of Thangarajah et al. [27,29] to incorporate preference information into summary information and we present algorithms to utilize this information to satisfy the user's preferences. It should be noted that our choice for the annotation of the goal-plan tree has been guided by the design of the Jadex platform [21] in which we have implemented our approach. The goal-plan tree mechanisms that we use has also been used by Thangarajah et al. for the agent system JACK [4].

It should also be noted that the scope of our work is the goal-plan tree, and in particular the choices to be made within it. Hence we do not consider other aspects of the BDI cycle, such as events and belief updates.

## 3 Preference specification

In this section we discuss the specification of the user preferences. These are expressed in a formal language using information that is extracted from the agent system. In short, we enable the developers of an agent system to annotate the goals and plans of the agent. After that the annotated information is processed and a summary of this information is presented to the user. The user can express preferences in terms of this summary information.

We note here that the role of the specification language is to allow the designers to specify preferences in a natural way, retaining a formal structure that allows reasoning over them. The language is not the 'most' natural nor the 'most' expressive, but is sufficient to illustrate our approach to how they can be included in the reasoning mechanisms.

Our preference specification therefore consists of two parts: a method of annotating the goals and plans of an agent system and a formal language in which the user can express preferences.

We introduce the notion of a $property$ of a goal which can be thought of as a relevant effect of the achievement of a goal. For example, a goal $G$ of booking a holiday may have a property called $payment$ which specifies the payment method used. Any plan that achieves $G$ by paying for the holiday with a credit card will result in the value $credit$ being assigned to this property. Similarly, an alternative plan may assign the value $debit$ for $payment$. This means that the values of the property $payment$ for goal $G$ is the set $\{credit, debit\}$.

The preference language we use allows the user to specify preferences over the possible values of these properties. For example, the statement "I would prefer for payment to be made via credit" states the preference for value $credit$ rather than $debit$ for the $payment$ property. The user can also specify preferences over the resource usage of a particular goal. Resource usage is specified in terms of an amount and comparative operator (e.g., at least, at most) for a particular resource type, such as "I prefer to spend at most $100 on transport".

The user then specifies preferences in terms of properties of goals and their values. *This means that the user does not need to know the details of how the goal is achieved in order to specify preferences.* For example, for the above user, it is sufficient to know that paying for a holiday can be done by credit or debit card; it is not necessary to know that in order to pay, a mode of transport and accommodation must be selected.

We annotate the goals and plans of the agent system with additional information which is then automatically propagated to parent goals and plans. This can also be seen in the example given earlier in this section, as the possible values of the property $payment$ of $G$ can be computed using the information in its plans. These annotations are then presented to the user who can then specify his or her preferences in a preference language.

In Sect. 3.1 we describe techniques for annotating and propagating information using the notion of summary information [27,29]. In Sect. 3.2 we describe a preference language based on the language $\mathcal{LPP}$ [1,2].

### 3.1 Summary information

We follow a similar approach to the one taken by Thangarajah et al. [27,29], in that goals and plans are augmented with resource and effect summaries to detect and resolve resource conflicts and to facilitate the merging of common subgoals. In this section we introduce *property summaries* for annotating relevant characteristics of plans and goals.

Figure 2 shows an example goal-plan tree of a goal of booking a holiday (HOLIDAYGOAL), which consists of two subgoals for booking accommodation (ACCOMMODATIONGOAL) and booking transport to the destination (TRANSPORTGOAL).

We annotate nodes in the goal-plan tree with the required information and automatically propagate this information to other nodes in the tree at compile time. The goal-plan tree in Fig. 2 shows both the programmer-specified properties $P^{PS}$ and resources $P^{R}$ and the propagated annotations of the summary information (property summaries $PS$ and resource summaries $RS$). It also serves as our motivating example.

We use resource summaries here as we also allow the user to specify preferences based on resources. A resource summary [29] contains the *necessary* and *possible* resource usage

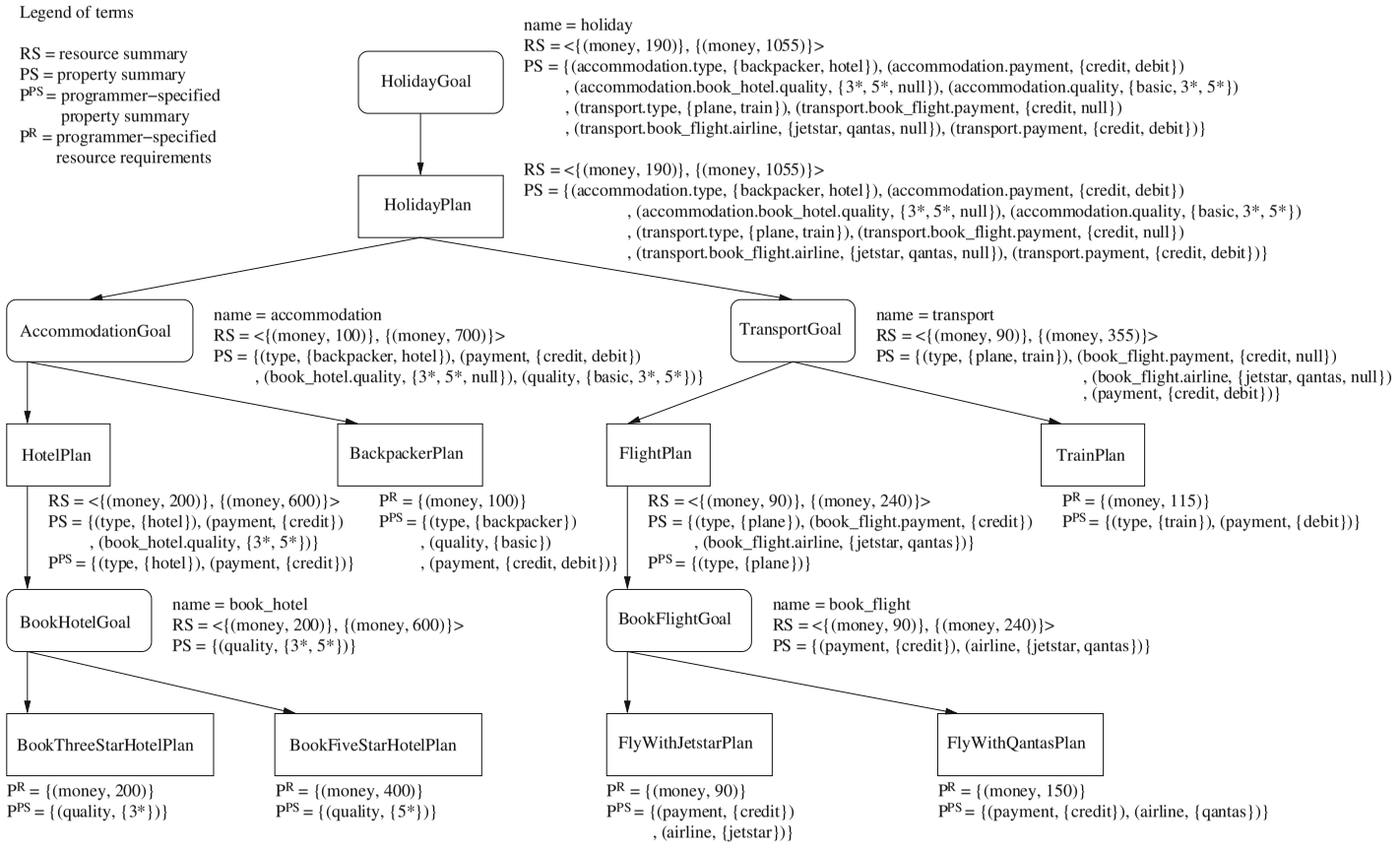


**Fig. 2** Goal-plan tree example with summary information

for a node ($\langle N, P \rangle$). Necessary resources are those that are used irrespective of the goal-plan choice and possible resources are those that may be needed in the worst case where plans may fail and alternatives are tried. Furthermore, Thangarajah et al. make a distinction between consumable resources and reusable resources that respectively can be used once and repeatedly. In this work we focus solely on consumable resources. For example, the resource *money* used by various plans in Fig. 2 is a consumable resource.

We note that in the example illustrated in Fig. 2, the possible resources for the BOOKHOTEL goal is $600, which is based on the assumption that if booking one particular hotel fails, the money is not refunded. This is not always the case in reality. In fact, the original design for this example contained a PAYFORHOTEL subgoal which was executed after a particular hotel was selected, however for illustration purposes we have simplified it here, sacrificing intuitiveness for presentation.

*Property summaries*

We now describe how property information can be summarized. We define a *property summary* $PS$ of a node $N$ (denoted as $PS_N$) in the goal-plan tree, that summarizes the properties of the subtree rooted at $N$, as follows. A *property* is of the form *(goalpath.name, values)* where *goalpath* denotes the path to $N$ in the goal-plan tree, *name* is the name of the property, and *values* is a set of values. A *property summary* is a non-empty set of properties. For a property $p =$ *(goalpath.name, values)*, we denote *name* by $name(p)$ and *values* by $values(p)$.

The reason that *values* must be non-empty is that the intended meaning of a property $p$ in the property summary of a node $N$ is that upon successful execution of $N$, the value of $p$ is exactly one of the set *values*. For example, upon successful execution of the node ACCOMMODATIONGOAL the value of the property *type* is exactly one of $\{backpacker, hotel\}$. Clearly this requires the values to be non-empty.

A *goalpath* contains the human-readable names of the goals, separated by dots, that are encountered on the path from the root goal to the property's node, excluding the name of the root goal node. We will also use the term *goalpath from G* to indicate the goalpath from a given goal $G$ in the goal-plan tree rather than the root, and refer to $goalpath.name$ where *goalpath* is the goalpath from a given goal $G$ as $localname(p, G)$. For example, the property summary of the node ACCOMMODATIONGOAL contains a property called $book\_hotel.quality$. This property has $book\_hotel$ as goalpath from ACCOMMODATIONGOAL and $quality$ as the property name, and its localname is $book\_hotel.quality$. Alternatively, we say that $book\_hotel.quality$ refers to the property $quality$ of the goal called $book\_hotel$.

We use the value $null \in values$ to indicate that the property could receive no value. For example, the goal TRANSPORTGOAL has two plans, FLIGHTPLAN and TRAINPLAN, where the former has the property $book\_flight.airline$ but the latter does not. Hence the property $book\_flight.airline$ of TRANSPORTGOAL will have the value *null* if TRAINPLAN is successfully executed.

Note that we allow properties to have more than one value. For example the property *payment* of BACKPACKERPLAN has values $\{credit, debit\}$. If the user does not prefer a particular value, the resulting value of the property is irrelevant.

### *3.1.1 Goal-plan tree node annotation*

The nodes in the goal-plan tree are annotated with programmer-specified information and computed summary information as we describe below.

*Programmer-specified information*

Each goal-node is annotated with the programmer-specified $goalname$. For each plan in the plan library, we require that a programmer specifies the resources required by that plan ($P^{\mathcal{R}}$), similar to the work of Thangarajah et al. [29], and additionally a property summary ($P^{PS}$). The corresponding plan-node in the goal-plan tree is annotated with this programmer-specified information.

For example, in Fig. 2, goal BOOKFLIGHTGOAL is annotated with its name, and its plans FLYWITHJETSTARPLAN and FLYWITHQANTASPLAN are both annotated with the sets of resources and properties required by the plan. Note that it is sufficient to provide this information just for the lowest level plans (i.e. the leaves of the goal-plan tree) as the information is propagated up in the goal-plan tree. We do however allow annotation for all plan-nodes, such as HOTELPLAN which has both programmer-specified and computed information. HOTELPLAN has two programmer-specified properties, $type$ and $payment$, and one property called $book\_hotel.quality$ which has been computed from its subgoal BOOKHOTELGOAL.

*Computed information*

Using the programmer-specified information above, we compute the resource summaries using the techniques described by Thangarajah et al. [29], and the property summaries using the techniques we develop in this work. Each node is annotated with this information.

An alternative approach is to explicitly attach a set of properties to any goal in the goal-plan tree, such as $properties = \{payment, airline\}$, to indicate the properties of the goal BOOKFLIGHTGOAL. However, this would require the agent designer to ensure that such properties are compatible with the plans used to achieve the goal. As the relevant properties can be inferred from a bottom-up evaluation of the goal-plan tree, it seems simpler to annotate only the plans in the plan library, and then use an automated process to propagate this information upwards. We also annotate each goal with a programmer-specified name which is used when propagating information upwards. These names do not have to be known when annotating the plans, as they are only used in the propagation process.

### *3.1.2 Propagation rules*

We now present propagation rules that formalize how the annotated information of a node and its child nodes can be used to compute the property summary of that node. The rules vary for goal and plan nodes.

*Plan node*

We start with the rule for computing the property summary of a plan. The properties of a plan are its programmer-specified ones combined with the properties of all the subgoals of the plan, if any. This is because when a plan is successfully executed, the properties of its subgoals will have received a value and we therefore regard them as part of the properties of that plan. We prepend the names of the properties of subgoals with the human-readable name of that goal to form part of the goalpath to identify from which subgoal the property has come from. For example, the subgoal ACCOMMODATIONGOAL of HOLIDAYPLAN has a property $(type, \{backpacker, hotel\})$ and the property summary of HOLIDAYPLAN contains a property called $accommodation.type$ with values $\{backpacker, hotel\}$ to distinguish it from other properties called $type$, such as the one from TRANSPORTGOAL.

**Definition 1** (*Propagation to a plan node*) Let $P$ be a plan node with programmer-specified property summary $P^{PS}$, and let $Goals(P)$ be the set of subgoals of $P$. Furthermore, let $G \in Goals(P)$ be a subgoal of $P$ with name $name_G$ and property summary $PS_G$. We define the property summary $PS_P$ of $P$ as:

$$PS_P \equiv_{def} P^{PS} \cup \bigcup_{G \in Goals(P)} \{(name_G.n, v) \mid (n, v) \in PS_G\}.$$

As an example, consider the plan HOLIDAYPLAN with subgoals ACCOMMODATIONGOAL and TRANSPORTGOAL. The programmer-specified property summary $P^{PS}$ is empty (i.e. there is nothing specific to this plan that has been added by the programmer), and so hence the properties of this plan are solely computed using the properties of its two subgoals. The properties of each subgoal are respectively prepended with their names *accommodation* and *transport* and all these properties together form the property summary of HOLIDAYPLAN.

*Goal node*

We now turn to the rule for computing the property summary of a goal. Recall that a goal has no programmer-specified properties. We compute the property summary of a goal based on the property summaries of its plans. Since we do not know which plan will be chosen for the goal at run time, we include the properties of each plan in the property summary of the goal.

We first define an operator $\oplus$ for merging two property summaries that adds the value *null* to the set of values of a property when a plan assigns at least one value to that property but a different plan does not. The assumption that we make here is that when two plans have a property with the same name then they are taken to refer to the same effect of the goal. For example, when the goal BOOKFLIGHTGOAL has two plans FLYWITHJETSTARPLAN and FLYWITHQANTASPLAN that both have a property called *airline* then we compute for the goal a property with the same name and with the values of each of these plans (i.e., a property *airline* with values $\{jetstar, qantas\}$). The operator $\oplus$ is defined as follows:

$$\begin{aligned} PS_1 \oplus PS_2 \equiv_{def} & \{(n, v_1 \cup v_2) \mid (n, v_1) \in PS_1 \wedge (n, v_2) \in PS_2\} \\ & \cup \{(n, v_1 \cup \{null\}) \mid (n, v_1) \in PS_1 \wedge (n, v_2) \notin PS_2\} \\ & \cup \{(n, v_2 \cup \{null\}) \mid (n, v_1) \notin PS_1 \wedge (n, v_2) \in PS_2\} \end{aligned}$$

In addition to properties that were obtained by merging properties with exactly the same name (e.g., *type* of BACKPACKERPLAN and HOTELPLAN for ACCOMMODATIONGOAL), possibly with the value *null* added, we also merge properties by assuming that properties of different plans with the same name but with a different goalpath represent the same characteristic of the goal. For example, we introduce a property *quality* for ACCOMMODATIONGOAL by merging the values of *quality* of BACKPACKERPLAN and *book_hotel.quality* of HOTELPLAN. The presence of both *quality* and *book_hotel.quality* for ACCOMMODATIONGOAL enables us to specify preferences at the level of accommodation in general and of hotels in particular.

We define a set $\mathcal{N}$ that contains the property names which have this characteristic as follows. We have $name \in \mathcal{N}$ iff $\exists P_1, P_2 \in Plans(G)$ such that $name \in PS_{P_1}$ and $goalpath.name \in PS_{P_2}$ where *goalpath* is the goalpath from $G$ and *goalpath* is not empty.

We also define a set $\mathcal{V}_{name}$ that contains the possible values of a property $name \in \mathcal{N}$. For each $v \in \mathcal{V}_{name}$ it holds that either

(1) for some plan $P$ of the goal and for some property $p \in PS_P$, we have that $name(p) = name \wedge v \in values(p)$, or
(2) $v = null$ when there exists a property summary $PS_P$ that does not contain a property $q$ such that $name(q) = name$.

The first condition is used to include the values of properties with that name. The second condition is used to include the value *null* when there is at least one plan that does not have a property with that name.

**Definition 2** (*Propagation to a goal node*) Let $G$ be a goal node, let $PLANS(G)$ be the set of plans for $G$ and let $P \in Plans(G)$ be a plan with property summary $PS_P$. We define the property summary $PS_G$ of $G$ as follows

$$PS_G \equiv_{def} \{(localname(p, G), \mathcal{V}_{name}) \mid localname(p, G) \in \mathcal{N}\} \cup \bigoplus_{P \in Plans(G)} \{p \mid p \in PS_P \wedge localname(p, G) \notin \mathcal{N}\}$$

For example, consider the AccommodationGoal with plans BackpackerPlan and HotelPlan with the following property summaries

$$\{(type, \{backpacker\}), (quality, \{basic\})\}$$

and

$$\{(type, \{hotel\}), (book\_hotel.quality, \{3^*, 5^*\})\}.$$

Note that here $\mathcal{N} = \{quality\}$. This means that we obtain the following properties for AccommodationGoal

$$\{(type, \{backpacker, hotel\}), (book\_hotel.quality, \{3^*, 5^*, null\}), (quality, \{basic, 3^*, 5^*\})\}$$

We observe that the values of *type* are merged and that the property *quality* results from the merge of *quality* and *book_hotel.quality* of respectively BackpackerPlan and HotelPlan. Furthermore, observe that the value *null* is added to the values of *book_hotel.quality* as this property could receive no value when the backpacker alternative is chosen.

The user specifies preferences in terms of the information in the summaries of the root node (top-level goal) of the goal-plan tree. The user therefore does not need to know the structure of the goal-plan tree and the goal-plan tree can be used by multiple users as preferences are specified separately from it. In Sect. 3.2 we will show how this information can be used to express the preferences.

### 3.2 Preference language

The preferences we wish to express are concerned with the values of properties and the resource usage of goals. Examples include:

– "I prefer to minimize the money spent on accommodation"
– "I prefer to fly rather than travel by train"
– "If the accommodation is a 5* hotel, I prefer to travel with Jetstar"

Our preference language is based on the language $\mathcal{LPP}$ [1,2]. Following the structure of $\mathcal{LPP}$, we use *basic desire formulas* to represent basic statements about the preferred situation, *atomic preference formulas* to represent an ordering over basic desire formulas and *general preference formulas* to express atomic preference formulas that are optionally subjected to a condition. We introduce the class of *conditions* that allow us to specify conditions with regard to information collected at run time.

The preferences of a user are specified as a set of general preference formulas.

**Definition 3** (*Basic Desire Formula*) A *basic desire formula* is either

1. the *name* of a goal property and a desired or an undesired *value*, expressed as $name = value$ or $name \neq value$, where *value* is not *null*,[2]
2. the predicate *minimize(resource)*, to express that the usage of *resource* should be minimized,[3]
3. the predicate *usage(resource, amount, comparator* $\in \{<, \leq, =, \geq, >\}$) to express that the usage of the *resource* should be according to the *comparator* and the *amount* (e.g., $usage(r, 500, \leq)$ implies at most 500 units of resource $r$ should be used), or
4. a predicate preceded by a goal name *goal* (e.g. *goal.minimize(resource)*).

If $\varphi_1, \ldots, \varphi_n$, $n \geq 2$, are of the form $name = value$ or $name \neq value$, then $\varphi_1 \wedge \cdots \wedge \varphi_n$ (conjunction) and $\varphi_1 \vee \cdots \vee \varphi_n$ (disjunction) are also basic desire formulas.

The first type of basic desire formulas can be used to express basic statements such as $transport.type = train$ or $accommodation.payment \neq credit$. The predicates $minimize$ and $usage$ can be used to express preferences about resource usage, such as $minimize(money)$ and $usage(money, 500, \leq)$. These predicates can also be prefixed with a goal name and we will explain later the purpose of these types of basic desire formulas. Lastly, we allow conjunctions and disjunctions of multiple basic desire formulas with a *name* and a desired or undesired *value*.

Basic desire formulas can be ordered in atomic preference formulas to express preferences in which a basic desire formula is preferred over another.

**Definition 4** (*Atomic Preference Formula*) Let $v_{min}$ and $v_{max}$ be numeric values such that $v_{min} < v_{max}$. An *atomic preference formula* is a formula

$$\varphi_0\ (v_0) \gg \cdots \gg \varphi_n\ (v_n),\ n \geq 0,$$

where $\varphi_i$ is a basic desire formula and $v_{min} \leq v_i \leq v_{max}$ and $v_i < v_j$ for $i < j$. If $\varphi_i$ is a predicate, then $\varphi_i$ can only be used when $i = n = 0$ (i.e., we do not allow predicates together with other basic desire formulas).

Note that we have dropped the constraint for $\varphi_0$ to have $v_0 = v_{min}$, in contrast to Bienvenu et al. [2]. We will explain later the reason for this modification. We use $v_{min} = 0$ and $v_{max} = 100$ in our work. We note that these numerical values (i.e., $v_0, \ldots, v_n$) in these atomic preference formulas are also present in the language $\mathcal{LPP}$. These values are used when evaluating the preferences. We present our method of evaluation in Sect. 4.1.2 but, in short, a lower value means more preferred than a higher value. An example of an atomic preference formula is

$$transport.type = plane\ (0) \gg transport.type = train\ (100),$$

[2] The value *null* serves a particular purpose that we have explained in Sect. 3.1.

[3] This predicate was included because the resource usage does not always need to be minimized when other preferences are taken into consideration.

which indicates that transport by plane is preferred to transport by train. Consider now the goal of booking transport with two plans that respectively book transport by plane and by train. With regard to this preference formula, the first of these plans will result in the value 0 (as it brings about $transport.type = plane$) whereas the second plan will result in the value 100 (as it brings about $transport.type = train$). We will discuss these numerical values in more detail in Sect. 4.1.2.

For simplicity we have constrained the use of basic desire formulas with predicates to atomic preference formulas with only one basic desire formula. As a result the following is *not* a valid atomic preference formula:

$$transport.type = plane\ (0) \gg minimize(money)\ (100)$$

We now define conditions which will be used later to define general preference formulas.

**Definition 5** (*Condition*) A *condition* is

1. the *name* of a goal property and a desired or an undesired *value*, expressed as $name = value$ or $name \neq value$, or
2. the predicate *success(goal)*, *failure(goal)*, or *used(goal, resource, amount)*, where *goal* is the name of a goal, $resource$ is the name of a resource and $amount \geq 0$ is a numerical value.

If $\varphi_1, \ldots, \varphi_n$, $n \geq 2$, are conditions then $\varphi_1 \wedge \cdots \wedge \varphi_n$ (conjunction) and $\varphi_1 \vee \cdots \vee \varphi_n$ (disjunction) are also conditions.

Some examples are $failure(book\_flight)$ and $success(transport)$. A condition, such as $failure(book\_flight)$, can be used to express preferences such as, "If I'm unable to travel by plane, then I prefer ...".

We now define general preference formulas to express atomic preference formulas that are optionally preceded by a condition.

**Definition 6** (*General Preference Formula*) A *general preference formula* is

1. an atomic preference formula, or
2. $\gamma : \Psi$, where $\gamma$ is a condition and $\Psi$ is an atomic preference formula.

Recall that the user preferences are specified as a set of general preference formulas. We can express the examples given in the beginning of this section as the following general preference formulas

$$accommodation.minimize(money)\ (0)$$
$$transport.type = plane\ (0) \gg transport.type = train\ (100)$$
$$accommodation.type = hotel\ \wedge\ accommodation.quality = 5* : book\_flight.airline = Jetstar\ (0)$$

## 4 Reasoning about preferences

We can identify two types of decisions that an agent needs to make: for a goal, an agent can select one of the plans and for a plan, an agent can choose the order in which to pursue the subgoals, if any, unless the order is determined by the structure of the plan. Both of these decisions can influence to what extent the user preferences will be satisfied. In this section we describe algorithms that utilize the user preferences and the summary information in the goal-plan tree to guide these decisions.

### 4.1 Plan selection for a goal

When the agent needs to select a plan for a goal, our approach is to express numerically how well a plan satisfies the preference formulas. We can then sort the plans from most to least preferred and attempt the plans in that order to achieve the goal.

#### *4.1.1 Preferences over resource usage*

Before we define how we evaluate preference formulas, we first discuss the differences between preferences that are concerned with goal properties and preferences that are concerned with the resource usage of a goal. These preferences differ in the sense that goal properties and their possible values are more precisely known to the agent than the resource usage of a goal. The latter depends on the possibly unsuccessful execution of plans in the subtree rooted at that goal whereas the value of a goal property must be one of the values in the known set of values.

Recall that the purpose of preferences is to guide the agent to make a rational decision. Since the actual resource usage of a node in the goal-plan tree is not known to the agent, we need to use estimates of the amount that will be used. The resource summaries of a node only contain the amounts that will necessarily and possibly be used. This is the only information the agent has with regard to the resource usage of a node. We argue that the use of resource estimates can lead to rational decisions as the resource usage of a plan or goal can be learned from previous executions. However, we do not prescribe how this must be done, and as below, we provide a flexible mechanism for this.

One such mechanism is the function `k-estimate` to compute the estimated resource usage of a particular resource of a plan.

**Definition 7** (*k-estimate*) Let $(nec_i, pos_i)$ be the necessary and possible usage of a particular resource for a plan $P_i$. We define the *k-estimate* of $P_i$ as $e_i = nec_i + k \cdot (pos_i - nec_i)$ where $0 \leq k \leq 1$.

The value of $k$ can be set for each plan and it is related to the expected failure rate of the plan, including the execution of its subgoals. We emphasize that the estimate serves to guide the agent and that this does not mean the agent is able to execute the plan with a resource usage that is close to the estimate.

#### *4.1.2 Evaluating preference formulas*

The method of evaluating formulas of our preference language is based on the semantics of the language by Bienvenu et al. For each class of preference formulas, we define an evaluation function $w$ that assigns a value $v_{min} \leq v \leq v_{max}$ to a formula of that class, where a lower value means more preferred. We evaluate preference formulas only for plan selection for a goal and we therefore evaluate a formula for a given goal $G$ and a plan $P_{eval}$ of $G$. We include $G$ in the evaluation of a preference formula because the evaluation of some basic desire formulas for a plan depends on the summary information of other plans of $G$. For example, the basic desire formula $minimize(resource)$ is satisfied for a plan if its resource usage of $resource$ is lowest compared to all other plans for that goal.

Compared to the work of Bienvenu et al., our work differs in the evaluation of our basic desire formulas and the evaluation of our introduced class of conditions. Otherwise, we follow the semantics of Bienvenu et al., apart from the adaptation of general preference formulas to

utilize conditions. Note that we only present a method of reducing a preference formula to a numerical value and not some rigorous formal semantics that describe when a formula is true in a model. These numerical values are utilized at run time to help the agent to make its decisions and we have left the development of formal semantics as part of future work.

It should be noted that the use of numerical values provides a means of comparing preferences. Ultimately either a preference is satisfied or it is not, but in the intermediate computations, it can be very useful to be able to provide a means of relative comparison, which the numerical values provide.

*Basic desire formula*

The following definition of evaluation of basic desire formulas consists of several cases which we explain in more detail soon.

**Definition 8** (*Basic Desire Evaluation*) Let $\varphi$ be a basic desire formula, let $G$ be a goal and let $Plans(G) = \{P_1, \ldots, P_n\}$. Furthermore, let $PS_i$ and $R_i$ respectively be the property summary and resource summary of $P_i$, and let $PS_{eval}$ be the property summary of $P_{eval}$. We define $w(\varphi, G, P_{eval})$ as follows

- if $\varphi$ is $name = value$ then $w(\varphi, G, P_{eval}) = v_{min}$ iff there exists a property $p \in PS_{eval}$, such that $value \in values(p)$ and $name$ is equal to either
  (1) $name(p)$ or $name(p)$ prepended with the goalpath from the root node to $G$, excluding the name of the root goal node, or
  (2) $name(p)$ with the goalpath to $G$ prepended, if there exists a plan $P_i \neq P_{eval}$ with a property $q \in PS_i$ such that $q$ has no goalpath and $name(q) = name(p)$.
  (3) $name(p)$ prepended with a goalpath from the root node to $G$, excluding the name of the root goal node and one or more of the goalnames of $G$ and parent goals.[4]

  Otherwise, $w(\varphi, G, P_{eval}) = v_{max}$.
- if $\varphi$ is $name \neq value$ then $w(\varphi, G, P_{eval})$ is as above for $(name = value)$ with the requirement that $value \notin values(p)$.
- if $\varphi$ is $\psi_1 \wedge \cdots \wedge \psi_m, m \geq 2$ then

$$w(\varphi, G, P_{eval}) = max(\{w(\psi_j, G, P_{eval}) \mid 0 \leq j \leq m\})$$

- if $\varphi$ is $\psi_1 \vee \cdots \vee \psi_m, m \geq 2$ then

$$w(\varphi, G, P_{eval}) = min(\{w(\psi_j, G, P_{eval}) \mid 0 \leq j \leq m\})$$

- if $\varphi$ is $minimize(resource)$ then $w(\varphi, G, P_{eval}) = v_{min}$ iff $P_{eval} \in S$, and $w(\varphi, G, P_{eval}) = v_{max}$ otherwise, where $S$ is a set of plans computed by one of the following procedures.[5]
  - `min_nec_pos` Let $(nec_i, pos_i)$ be the necessary and possible resource usage of $resource$ for $P_i$. Let $S$ contain each plan $P_i$ that satisfies
    (1) there is no $P_j, i \neq j$ such that $nec_j < nec_i$,
    (2) if multiple plans satisfy (1), we include in $S$ only those for which it additionally holds that there is no $P_j, i \neq j$ such that $pos_j < pos_i$.

[4] Conditions (2) and (3) are used for properties that were merged using $\mathcal{N}$ as described in Sect. 3.1.2. We provide a detailed example in Sect. 5.4

[5] The procedure to be used is determined by the designer prior to execution. Other reasonable choices for this procedure can be used if the designer so wishes.

  - `min_estimate` Let $e_i$ be the $k$-`estimate` of a plan $P_i$. Then $S$ contains each plan $P_i$ for which there is no $P_j, i \neq j$ such that $e_j < e_i$.

- if $\varphi$ is $usage(resource, amount, comparator)$ then $w(\varphi, G, P_{eval}) = v_{min}$ iff $P_{eval} \in S$, and $w(\varphi, G, P_{eval}) = v_{max}$ otherwise, where $S$ is a set of plans computed using the procedure corresponding to the $comparator \in \{<, \leq, =, \geq, >\}$ that is used.
  Let $(nec_i, pos_i)$ be the necessary and possible resource usage of $resource$ for $P_i$, and $e_i$ be the $k$-`estimate` for $P_i$.
  - $<$ and $\leq$ $amount$: Then $S$ contains all plans $P_i$ such that $pos_i < amount$ ( resp. $\leq$). If no such plans exist, then $S$ contains all plans $P_i$ such that $e_i < amount$ (resp. $\leq$).
  - $>$ and $\geq$ $amount$: Then $S$ contains all plans $P_i$ such that $nec_i > amount$ (resp. $\geq$). If no such plans exist, then $S$ contains all plans $P_i$ such that $e_i > amount$ (resp. $\geq$).
  - $=$ $amount$: Then $S$ contain all plans $P_i$ such that $|e_i - amount|$ is lowest (i.e., $e_i$ is closest to $amount$).

- if $\varphi$ is the predicate *minimize* or *usage* preceded by a goal name *goal* then the aforementioned evaluation of these predicates are used iff *goal* is equal to the name of $G$ or equal to the name of one of the parent goals of $G$. Otherwise, $w(\varphi, G, P_{eval}) = v_{max}$.

We note that for the evaluation of $minimize(resource)$ we take into account whether a plan has previously failed. For example, if plan $P_1$ requires $a_1$ of a particular resource, $P_2$ requires $a_2$ of that resource, and $P_3$ requires $a_3$, where $a_1 < a_2 < a_3$, then $S = \{P_1\}$ prior to the execution of any plans but $S = \{P_2\}$ after the execution of $P_1$ has failed (i.e., the failed plan $P_1$ is no longer considered when computing $S$). A detailed example of this can be found in Sect. 5.3.

Let us take a closer look at these cases using some examples.

- If a basic desire formula $\varphi$ has the form $name = value$ or $name \neq value$ then we analyze the property summary for a matching property.
  Let $\varphi_1$ be $type = hotel$ and let $p$ be

$$(type, \{backpacker, hotel\}) \in PS_{eval}.$$

  The basic desire formula $\varphi_1$ is *satisfied* for this property summary due to the first condition in the definition: we have that $hotel \in \{backpacker, hotel\}$ and that $type$ is equal to $name(p) = type$. Note that the situation described by the first condition of the definition does not occur in Fig. 2 but this condition is necessary for properties that belong to plans of the top-level goal. The situation that we describe next is more common because it involves the goalpath and as we can see in Fig. 2 most properties in property summaries have a prepended goalpath.
  Let $\varphi_2$ be $accommodation.type = hotel$ and let $p$ be

$$(type, \{hotel\}) \in PS_{eval}.$$

  Furthermore, let $PS_{eval}$ be the property summary of a plan for the goal called $accommodation$. This example can be seen in Fig. 2 in the node HotelPlan. The basic desire formula $\varphi_2$ is *satisfied* for this property summary due to the first condition (second part) in the definition: we have that $hotel \in \{hotel\}$ and that $accommodation.type$ is equal to $name(p) = type$ with the goalpath ($accommodation$) prepended. When a user preference formula contains this basic desire formula we can see that the plan HotelPlan of AccommodationGoal satisfies this formula whereas the plan BackpackerPlan does not.

Let $\varphi_3$ be $accommodation.quality = 3^*$ and let $p$ be

$$(book\_hotel.quality, \{3^*, 5^*\}) \in PS_{eval}.$$

This example can be seen in Fig. 2 for the nodes AccommodationGoal and HotelPlan, the plan for which we are evaluating ($P_{eval}$). The user has specified a preference using the introduced property $accommodation.quality$. We observe that due to the presence of the property $quality$ in BackpackerPlan (see the discussion in Sect. 3.1), we are able to match $accommodation.quality$ to $book\_hotel.quality$ in HotelPlan. When evaluating $\varphi_3$ for AccommodationGoal we can see that the preference for 3* accommodation can possibly be accomplished when pursuing HotelPlan rather than BackpackerPlan.

– Let $\varphi_4$ be $minimize(money)$. According to our method of evaluation we can use either `min_nec_pos` or `min_estimate` to determine which plan or plans of a goal satisfy $\varphi_4$. Consider a goal $G$ with plans $P_1$ with resource summary $\langle\{(money, 200)\}, \{(money, 600)\}\rangle$, a plan $P_2$ with resource summary $\langle\{(money, 100)\}, \{(money, 100)\}\rangle$. This example can be seen in Fig. 2 for AccommodationGoal with HotelPlan and BackpackerPlan respectively.
  Let us use `min_nec_pos` to compute a set $S$ with plans that satisfy both criteria. Informally, $S$ contains all plans with the lowest necessary usage and in case there are several, we include in $S$ only those with the lowest possible usage. We can see that $S = \{P_2\}$ as $P_2$ has the lowest necessary resource usage and all other plans, in this case only $P_1$, have a higher necessary resource usage. This means that $\varphi_4$ is only satisfied when $P_{eval} = P_2$ but not when the formula is evaluated for $P_1$. More concretely, $\varphi_4$ is satisfied for the plan BackpackerPlan of AccommodationGoal but not for HotelPlan, where the former plan is indeed the cheaper alternative of both accommodation plans.
– Lastly, let $\varphi_5$ be $usage(money, 300, \leq)$. Consider again the goal AccommodationGoal and its plans HotelPlan and BackpackerPlan as specified in Fig. 2. We follow the procedure as defined for the comparative operator $\leq$ to compute a set $S$ of plans for which $\varphi_5$ is satisfied. The possible resource usage of money for HotelPlan is 600 whereas the possible resource usage of money for BackpackerPlan is 100 (as noted in the previous example). We observe that $100 \leq 300$ yet $600 \not\leq 300$ and therefore we obtain $S = \{Backpacker\,Plan\}$. Hence, $\varphi_5$ is satisfied for BackpackerPlan but not for HotelPlan.

*Atomic preference formula*

Recall that an atomic preference formula orders one or more basic desire formulas, each with an associated value. The value we wish to associate with an atomic preference formula $\Phi$ as a whole is the value of a satisfied basic desire formula in $\Phi$ with the lowest associated value.

**Definition 9** (*Atomic Preference Evaluation*) Let $\Phi = \varphi_0\ (v_0) \gg \cdots \gg \varphi_n\ (v_n)$, $n \geq 0$, be an atomic preference formula and let $G$ be a goal. We define $w(\Phi, G, P_{eval}) = v_i$ if there exists $\varphi_i$ such that $w(\varphi_i, G, P_{eval}) = v_{min}$ and there is no $j < i$ such that $w(\varphi_j, G, P_{eval}) = v_{min}$. Otherwise, $w(\Phi, G, P_{eval}) = v_{max}$.

The evaluation of conditions requires information about the execution of goals thus far. For this we define metadata

$$\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_S, \mathcal{M}_F, \mathcal{M}_R \rangle$$

to contain

- $\mathcal{M}_P$ pairs of a property name and its received value,
- $\mathcal{M}_S$ names of goals that have succeeded,
- $\mathcal{M}_F$ names of goals that have failed, and
- $\mathcal{M}_R$ a data structure with per goal and per resource the amount that was used. We use $\mathcal{M}_R^{goal}(resource)$ to denote the amount (possibly none) of $resource$ used thus far for the execution of $goal$.

*Conditional formula*

We can now define the evaluation of conditions using the above metadata which contains information collected at run time about the agent's execution.

**Definition 10** (*Condition Evaluation*) Let $\Phi$ be a condition and let metadata $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_S, \mathcal{M}_F, \mathcal{M}_R \rangle$. We define $w(\Phi, \mathcal{M})$ as follows

- if $\Phi$ is not a conjunction or disjunction, we define $w(\Phi, \mathcal{M}) = v_{min}$ iff one of the conditions below holds and $w(\Phi, \mathcal{M}) = v_{max}$ otherwise.
  - if $\Phi$ is $name = value$ then $(name, value) \in \mathcal{M}_P$
  - if $\Phi$ is $name \neq value$ then $(name, value) \notin \mathcal{M}_P$
  - if $\Phi$ is $success(goal)$ then $goal \in \mathcal{M}_S$
  - if $\Phi$ is $failure(goal)$ then $goal \in \mathcal{M}_F$
  - if $\Phi$ is $used(goal, resource, amount)$ then $\mathcal{M}_R^{goal}(resource) \geq amount$
- if $\Phi$ is $\varphi_i \wedge \cdots \wedge \varphi_n, n \geq 2$ then

$$w(\Phi, \mathcal{M}) = max(\{w(\varphi_i, \mathcal{M}) \mid 0 \leq i \leq n\})$$

- if $\Phi$ is $\varphi_i \vee \cdots \vee \varphi_n, n \geq 2$ then

$$w(\Phi, \mathcal{M}) = min(\{w(\varphi_i, \mathcal{M}) \mid 0 \leq i \leq n\})$$

We note that although conditions are generally evaluated to *true* or *false*, in the definition above they evaluate to *value* as this allows a comparison between different conditional preference formulas.

*General preference formula*

Lastly, we define the evaluation of the class of general preference formulas in which the user specifies its preferences.

**Definition 11** (*General Preference Evaluation*) Let $\Phi$ be a general preference formula and let $G$ be a goal. We define $w(\Phi, G, P_{eval}, \mathcal{M})$ as

- $w(\varphi_0 \gg \cdots \gg \varphi_n, G, P_{eval}, \mathcal{M}) = w(\varphi_0 \gg \cdots \gg \varphi_n, G, P_{eval})$ (i.e. when $\Phi$ is just an atomic preference formula)
- $w(\gamma : \Psi, G, P_{eval}, \mathcal{M}) = \begin{cases} v_{min}, & \text{if } w(\gamma, \mathcal{M}) = v_{max} \\ w(\Psi, G, P_{eval}), & \text{otherwise} \end{cases}$

Recall that in atomic preference formulas, we do not define $\varphi_0$ to have value $v_0 = v_{min}$. This allows preference formulas to override other formulas. For example, we can prefer

$$prop = val_1 \ (100)$$

and

$$\gamma : prop = val_2\ (50),$$

where $\gamma$ is a condition. If $\gamma$ is satisfied then a plan that satisfies $prop = val_2$ is preferred over a plan that satisfies $prop = val_1$.

Furthermore, the numerical values in the user preferences can be used to influence how important it is that certain properties are satisfied. Consider the following formulas that specify preferences with regard to properties of goal $goal_1$:

$$\varphi_1 : goal_1.prop_1 = val_1\ (0) \gg goal_1.prop_2 = val_2\ (100)$$
$$\varphi_2 : goal_1.prop_3 = val_3\ (50) \gg goal_1.prop_4 = val_4\ (75)$$

A plan $P$ that brings about the properties $goal_1.prop_1 = val_1$ and $goal_1.prop_3 = val_3$ will respectively result in the values 0 and 50 for preference formulas $\varphi_1$ and $\varphi_2$. As we will see later this results in the value 0 + 50 = 50 for $P$ if $\varphi_1$ and $\varphi_2$ are the only plans for $goal_1$. However, $P$ would result in the value $100+50 = 150$ if it would not bring about the property $goal_1.prop_1 = val_1$ and in the value $0 + 75 = 75$ if it would not bring about the property $goal_1.prop_3 = val_3$. As a lower evaluation value is more preferred than a higher evaluation value, we can see that not satisfying $goal_1.prop_1 = val_1$ incurs a higher 'penalty' than not satisfying $goal_1.prop_3 = val_3$.

We can now present our algorithm for computing the preferred order in which plans of a goal $G$ should be selected for execution. The input of this algorithm is the set of general preference formulas $\mathcal{F}$, the goal $G$ and metadata $\mathcal{M}$. The algorithm consists of the following steps:

- For each plan $P_i$ of $G$, compute a score $score_i$ which is the sum of the values of $w(f, G, P_i, \mathcal{M})$ for each $f \in \mathcal{F}$.
- Sort the plans by $score_i$ in non-decreasing order.

The output of this algorithm is an ordered list of the plans and the agent attempts the plans in that order. In case of plan failure, the next plan in the ordered list is attempted.

### 4.2 Order of subgoals of a plan

Recall that the other type of decision is to determine the execution order of subgoals of a plan, when their order is not fixed by design. Our approach is to infer the ordering of subgoals from the general preference formulas containing a condition. Consider the general preference formula

$$goal_1.prop_1 = value_1 : goal_2.prop_2 = value_2\ (0)$$

which can be read as "if $prop_1$ of $goal_1$ has received the value $value_1$ then I prefer $prop_2$ of $goal_2$ to receive $value_2$". To satisfy this preference, we should execute $goal_1$ before $goal_2$ to determine the value of $prop_1$. If its value is indeed $value_1$ then we can aim to satisfy the preferred value of $prop_2$ for $goal_2$. We wish to satisfy the user preferences as much as possible which is why the agent should, given a general preference formula $\gamma : \Psi$, execute the goals mentioned in $\gamma$ before the goals that are referred to in $\Psi$. We can determine these goal orderings at compile time by analyzing the preference formulas and the structure of the goal-plan tree.

Consider the following preference formula

$$book\_flight.airline \neq qantas : accommodation.type = hotel\ (0)$$

Based on our previous observations, we want to execute BOOKFLIGHTGOAL before ACCOMMODATIONGOAL to obtain the value of $book\_flight.airline$. By analyzing the goal-plan tree, we see that BOOKFLIGHTGOAL is part of the subtree rooted at TRANSPORTGOAL. Furthermore, both TRANSPORTGOAL and ACCOMMODATIONGOAL are subgoals of HOLIDAYPLAN. We can see that at HOLIDAYPLAN, the agent should therefore pursue the subgoal TRANSPORTGOAL before ACCOMMODATIONGOAL.

In general, for a plan with subgoals $G_0, \ldots, G_n, n \geq 0$, we want to determine if there are any pairs of goals $G_i$ and $G_j$, for some $i, j, i \neq j$, such that $G_i$ should be executed before $G_j$. To represent this information, we define an *ordering set* for each plan.

**Definition 12** (*Ordering Set*) An ordering set $\mathcal{O}_P$ of a plan $P$ is a possibly empty set of triples $(g_1, g_2, weight)$, where $g_1$ and $g_2$ are human-readable names of goals, expressing that $g_1$ should be executed before $g_2$, and $weight \geq 1$ is the number of preference formulas that have given rise to this goal ordering.

The value $weight$ determines which goal orderings take precedence over others when multiple formulas result in conflicting orderings. Goal orderings with a higher weight are considered to be important for satisfying possibly more preference formulas than those with a strictly lower weight.

In the following sections we present two algorithms: an algorithm for computing the ordering sets prior to an agent's execution (Sect. 4.2.1) and an algorithm for utilizing the ordering sets when determining the order of the subgoals of a plan (Sect. 4.2.2).

#### *4.2.1 Computing ordering sets*

Recall that we can determine the ordering of subgoals of a plan by analyzing the user preferences as described in Sect. 4.2. Each ordering constraint that can be found in a preference formula has the form "goal $G_1$ should be executed before goal $G_2$". For a given plan $P$ in the goal-plan tree we can analyze whether this ordering constraint influences the order in which the subgoals of $P$ should be executed to satisfy the preference.

- If $G_1$ and $G_2$ are found in a part of the goal-plan tree that is not related to $P$ (i.e., neither of the goals is part of the subtree rooted at $P$) then the ordering constraint can be ignored. The agent can influence the execution of the subgoals of $P$ but the decisions that are made here do not influence whether the preference regarding $G_1$ and $G_2$ will be satisfied.
- If either $G_1$ or $G_2$ is part of the subtree rooted at $P$ and the other goal is not then no constraint needs to be placed on the execution order of the subgoals of $P$. Similar to the previous case, the execution order of the subgoals of $P$ has no influence on whether this preference will be satisfied.
- If both $G_1$ and $G_2$ are part of the subtree rooted at $P$ then the execution order of the subgoals can influence whether the ordering preference is satisfied or not. For example, if $G_1$ and $G_2$ are direct subgoals of $P$ and we execute $G_2$ before $G_1$ then the preference of "$G_1$ before $G_2$" is not satisfied.

We will now analyze the third case in which both goals are part of the subtree rooted at $P$. We can distinguish three cases of preferred orderings of goals by looking at the position of a goal in the subtree rooted at $P$, whether the goal is a direct subgoal of $P$ or is it a subgoal deeper in the goal-plan tree.

(A) The execution of a direct subgoal can be preferred before another direct subgoal.

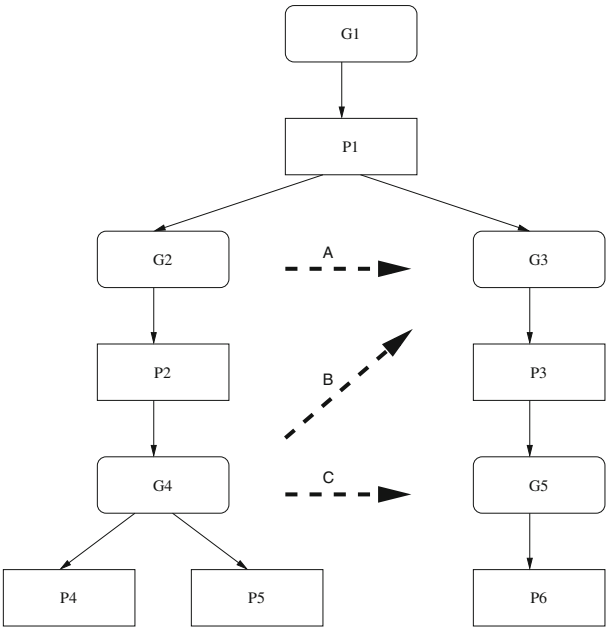


**Fig. 3** Three cases of goal orderings can be distinguished in user preferences. A *thick dashed line* from a goal node $G_i$ to $G_j$ indicates that $G_i$ should preferably be executed before $G_j$.

(B) The execution of a subgoal deeper in the goal-plan tree can be preferred before another direct subgoal.
(C) The execution of a subgoal deeper in the goal-plan tree can be preferred before another subgoal deeper in the goal-plan tree.

Figure 3 shows these three cases in thick dashed lines. Let us consider each of these cases to determine the information that should be put into the ordering set of plan $P_1$ as a result of this ordering preference.

(A) Each entry in the ordering set of $P_1$ specifies an ordering constraint over direct subgoals of $P_1$. In this case that information is readily available and no further computations are needed. Following the example in Fig. 3, we can add or update an entry in the ordering set of $P_1$ expressing that subgoal $G_2$ should be executed before subgoal $G_3$.
(B) In this case we have a subgoal that is not a direct subgoal of $P_1$ for which we are computing the ordering set. Before we can add or update the ordering set, we need to compute to which direct subgoal this deeper subgoal belongs. We can repeatedly retrieve the parent node of a node in the goal-plan tree until we arrive at a direct subgoal of $P_1$. We know that the deeper subgoal is part of the subtree rooted at $P_1$ so we know that the aforementioned situation must occur. When we have found the two direct subgoals of $P_1$ (i.e., the direct subgoal that is given and the one we computed), we can add or update an entry in the ordering set with the ordering constraint of two subgoals. In Fig. 3 the preference that subgoal $G_4$ should be executed before $G_3$ results in the ordering constraint that the subgoal $G_2$ should be executed before subgoal $G_3$.
(C) We follow the same approach as in case (B) but now for both goals as they are not direct subgoals of $P_1$. In Fig. 3 the preference of executing $G_4$ before $G_5$ results in the ordering constraint of executing $G_2$ before $G_3$.

Recall that an entry in the ordering set consists of an ordering constraint between two subgoals and a $weight$. The $weight$ is by definition the number of preference formulas that have given rise to this goal ordering. When we have determined the ordering constraint we also need to associate a $weight$ with this constraint. If the ordering set does not contain an entry with this ordering constraint we set the $weight$ to one. Otherwise we increment the weight of an existing entry in the ordering set. Note that is is possible to have more sophisticated mechanisms here, such as giving some preference formulas more weight than others, so that it is not just a simple count of preference formulas. Investigation of such weights and similar issues is an item of future work.

We have discussed how to compute the entries in an ordering set for a particular plan. For each ordering constraint in a preference formula we (1) determine the plan to which it belongs and (2) we then follow the above procedure to compute the entry of its ordering set.

How do we determine the plan (i.e. the aforementioned step 1) for which we need to update the ordering set? We have seen that there is only one case which is relevant for satisfying the user preferences: the case in which both goals in the ordering constraint are part of the subtree rooted at the plan. Given two ordering constraints, we are therefore looking at a plan that is an ancestral node of both goals in the goal-plan tree.

To be precise, we are looking for an ancestral plan node of both goals at which we can influence whether the ordering constraint is satisfied or not. Not all ancestral plan nodes are useful in this regard because at some ancestral plan nodes both goals of the ordering constraint belong to the same subtree underneath that plan node. For example, in Fig. 3 the ordering constraint C that goal $G_4$ should preferably be executed before $G_5$ can be satisfied by choosing the correct order of subgoals at plan $P_1$ but not at a (hypothetical and not pictured) plan node $P_0$ that is a parent of $G_1$.

From this analysis we can see that we are looking for the *nearest* ancestral plan node that is a parent node of both goals in the goal-plan tree. The ancestral plan node $P$ in the goal-plan tree is 'nearest' to goal nodes $G_i$ and $G_j$ in the sense that

(1) $P$ is an ancestral plan node of both $G_i$ and $G_j$, and
(2) $P$ does not have a descendant plan node for which (1) holds.

We can now present Algorithm 1 for computing the ordering set of each plan in the goal-plan tree. Algorithm 1 is part of the initialization phase (i.e., prior to execution of goals and plans of the agent) in which also the resource and property summaries are being propagated through the goal-plan tree.

#### *4.2.2 Utilizing ordering sets to order the subgoals of a plan*

The next algorithm, Algorithm 2, that we will present is used to decide the preferred order of subgoals of a plan based on its ordering set. Algorithm 2 can be executed in the initialization phase or at run time when the plan needs to instantiate its subgoals. Regardless of when the algorithm is executed, the algorithm does not take information collected at run time into account. This means the order of subgoals of plans is determined by analyzing the user preferences without information about the current situation. If the agent were able to take the current situation into account, it might prefer a different execution order of subgoals. We will elaborate on this further in our discussion of future work (Sect. 6).

The algorithm starts with an ordering of the subgoals, such as $g_1, g_2, g_3, g_4, g_5$, and it processes the elements of the ordering set in non-increasing order from highest to lowest weight. The algorithm does not enforce a particular ordering when processing elements with the same weight. The outcome of the algorithm is an ordered list of subgoals and the subgoals

**Algorithm 1** Computing ordering sets

**Require:** goal-plan tree $\mathcal{G}$ and set of preference formulas $\mathcal{F}$
Initialize ordering set $\mathcal{O}_P$ of each plan $P$ to $\emptyset$
**for each** formula $f \in \mathcal{F}$ such that $f$ is $\gamma : \Psi$ **do**
  $goals_{cond} \leftarrow$ all goal names present in $\gamma$ (for $name = value$ or $name \neq value$ we take the right-most goal name in goalpath of $name$)
  $goals \leftarrow$ all goal names present in $\Psi$ (for $name = value$ or $name \neq value$ we take the right-most goal name in goalpath of $name$)
  **for each** $g_{cond} \in goals_{cond}$ **do**
    **for each** $g \in goals$ such that $g \neq g_{cond}$ **do**
      $P \leftarrow$ nearest common ancestral plan of $g$ and $g_{cond}$ in $\mathcal{G}$
      $g_1 \leftarrow$ subgoal of $P$ and root of subtree with $g_{cond}$
      $g_2 \leftarrow$ subgoal of $P$ and root of subtree with $g$
      Insert $(g_1, g_2, 1)$ into $\mathcal{O}_P$ or increment weight $w$ of an existing entry $(g_1, g_2, w) \in \mathcal{O}_P$
    **end for**
  **end for**
**end for**

should be pursued in that order, e.g., $g_5, g_3, g_4, g_2, g_1$. Note that each goal ordering constraint (which is obtained from a preference formula of the user) results in an entry in the ordering set of the nearest ancestral plan node of those two goals. In Fig. 3 any ordering constraint with a goal from the left subtree ($G_2$ and $G_4$) and a goal from the right subtree ($G_3$ and $G_5$), such as "$G_2$ should preferably be executed before $G_5$", results in an entry in the ordering set of $P_1$. Similarly, in Fig. 2 any ordering constraint results in an entry in the ordering set of HOLIDAYPLAN as the structure of the goal-plan tree is similar to the one in Fig. 3, with a few additional plan nodes.

**Algorithm 2** Computing preferred order of subgoals

**Require:** goals $G_0, \ldots, G_n$ of plan $P$ and ordering set $\mathcal{O}_P$
$result \leftarrow$ an ordering of $G_0, \ldots, G_n$
$max \leftarrow$ highest weight present in $\mathcal{O}_P$
$prevs \leftarrow \emptyset$
**for** $lvl = max$ to 1 **do**
  $goals \leftarrow \{(g_1, g_2, w) \in \mathcal{O}_P \mid w = lvl\}$
  **for** $(g_1, g_2, w) \in goals$ **do**
    **if** moving $g_2$ to end of $result$ does not violate goal orderings in $prevs$ **then**
      $result \leftarrow result$ with $g_2$ moved to the end
      $prevs \leftarrow prevs \cup \{(g_1, g_2, w)\}$
    **end if**
  **end for**
  $lvl \leftarrow lvl - 1$
**end for**
**return** $result$

We will discuss Algorithm 2 using the following example. Consider a plan $P$ with subgoals $g_1, \ldots, g_5$ and the ordering set

$$\mathcal{O}_P = \big\{(g_1, g_2, 3), (g_1, g_3, 1), (g_2, g_1, 4), (g_3, g_4, 3)\big\}.$$

We use the notation $\psi_1 \rightarrow^{(g_i, g_j, w)} \psi_2$ to denote that the ordering $\psi_1$ was transformed into $\psi_2$ after processing $(g_i, g_j, w)$. The execution of the algorithm is as follows:

$g_1, g_2, g_3, g_4, g_5$
$\longrightarrow^{(g_2,g_1,4)}$ $g_2, g_3, g_4, g_5, g_1$ ($g_1$ moved to back, $(g_2, g_1, 4)$ added to $prevs$)
$\longrightarrow^{(g_1,g_2,3)}$ $g_2, g_3, g_4, g_5, g_1$ (ignored, would violate $(g_2, g_1, 4)$)
$\longrightarrow^{(g_3,g_4,3)}$ $g_2, g_3, g_5, g_1, g_4$ ($g_4$ moved to back, $(g_3, g_4, 3)$ added to $prevs$)
$\longrightarrow^{(g_1,g_3,1)}$ $g_2, g_3, g_5, g_1, g_4$ (ignored, would violate $(g_3, g_4, 3)$)

We have now obtained an ordering of subgoals that satisfies the goal orderings $(g_2, g_1, 4)$ and $(g_3, g_4, 3)$. The goal ordering $(g_1, g_2, 3)$ is ignored as it violates the earlier constraint that $g_2$ should be executed before $g_1$. The goal ordering $(g_1, g_3, 1)$ is ignored because it would violate $(g_3, g_4, 3)$ if $g_3$ was moved to the end of the list of goals. Note that it is possible to satisfy this constraint by putting $g_3$ between $g_1$ and $g_4$ but our algorithm does not support this operation. We refer the reader to Sect. 6 for a discussion on this issue.

It should be noted that our algorithm is not intended to guarantee a certain level of optimality, but to provide the user of the system with a systematic mechanism for indicating preferences. These are then used as a input to the decision-making process within the agent, so that the decisions made are in accord with the user's desires. As with many such mechanisms, this becomes a trade-off between the time spent in specifying preferences and the user's perception of the quality of the solutions found.

## 5 Implementation and case studies

We have implemented our preference system in the agent platform Jadex [21] and have tested it on a number of examples, including the holiday example discussed above. The implementation consists of around 3000 lines of code, which utilizes the `metagoal`, `metaplan`, `mastergoal` and `masterplan` features of Jadex as follows:

- We add a `metagoal` and a `metaplan` for meta-level reasoning on plan selection. Both have the predefined parameters `applicable`, which contains the plans to choose from, and `result`, which must contain the chosen plan after the `metaplan` has been executed. For every goal, the `metagoal` is instantiated and in its `metaplan`, we evaluate the alternative plans and we store the preferred plan in `result`.
- We add a `mastergoal` with a `masterplan` which run before all other goals and plans to (1) extract the goal-plan tree from the agent specification, to (2) annotate and propagate the additional information in the goal-plan tree and to (3) instantiate the root goal node. These tasks should ideally be integrated into Jadex and they should not be written by an agent programmer.
- We attach a status (`success`, `failure`, `active` and `default`) to each node to reflect the state of the goals and plans in Jadex. We use this information when making a decision for a goal. For example, a goal has plans $P_1$, $P_2$, and $P_3$ and $P_2$ is the most preferred plan but $P_2$ has previously failed so we return the second-most preferred plan. The status of each node is updated whenever a plan execution finishes.
- We update metadata $\mathcal{M}$ (see Sect. 4.1) whenever a plan execution finishes. For every successful plan, we store the values of its properties and all ancestral properties that are based on it (e.g., $accommodation.book\_hotel.quality$ and $accommodation.quality$). Note that the resource usage of a plan affects the resource usage of all its ancestral goals.

We use this implementation to present case studies to illustrate the features of our system as follows.

### 5.1 The example system

We implemented the holiday booking system outlined in Sect. 3.2 with the goal-plan tree show in Fig. 2. We present the user preferences mentioned here again for convenience:

"I prefer to minimize the money spent on accommodation."
$accommodation.minimize(money)$ (0)
"I prefer to fly rather than travel by train."
$transport.type = plane$ (0) $\gg$ $transport.type = train$ (100)
"If the accommodation is a 5* hotel, I prefer to travel with Jetstar."
$accommodation.type = hotel \wedge accommodation.quality = 5* :$
$book\_flight.airline = Jetstar$ (0)

We emphasise that our motivations for implementing this and similar examples is to illustrate the usefulness of our techniques, and not to perform a general evaluation of the utility of preference-based reasoning in intelligent agent systems (which would be a much more significant undertaking). Such an evaluation would require different groups of people with similar skills in agent programming to use a platform with and without the preference addition and check the time, quality and results of the systems. This type of evaluation is clearly out of the scope of this paper. Another type of evaluation might be to see how much extra time the calculation of the preferred plan will take over just taking a default choice. This, of course, depends on the amount of information added and the number and size of the plans. Our implementation has shown that the extra reasoning needed to support our techniques takes negligible time. This is due to the compositional, bottom-up method we use to construct preferences. It means that while constructing a plan at each choice point just a few extra operations are performed and the overhead is restricted to a linear number of extra calculations with respect to the number of decision points (i.e. the size of the goal-plan tree). Given the small differences in performances, it did not seem worthwhile making foraml measurements of the overhead involved.

Our aim with the case studies below, then, is not to perform a detailed evaluation of preference-based reasoning in agent systems, but to show how our techniques work on some concrete examples, and to make as precise as possible the relationship between the user's preferences and the output of the system. In other words, we provide some insight into the question of whether the given solutions are what the user really wants.

We executed the system with and without preference based reasoning. When preferences were not used or when no preferences were available, the agent randomly selected a plan for a goal and pursued the subgoals of a plan in an arbitrary order. The user preferences were therefore only satisfied when the agent happened to make the right decisions during executions. When we included our reasoning algorithms, the agent booked backpacker accommodation (as cheapest alternative) and a flight with either airline as expected. In case the plan for backpacker accommodation failed, the agent booked a 3* hotel. When this plan also failed, the 5* hotel was selected and due to the third preference formula, the agent selected the plan to fly with Jetstar rather than Qantas. Furthermore, the accommodation goal was pursued before the transport goal to make this possible.

The above findings are unsurprising and as expected. In order to illustrate the more interesting aspects of our preference reasoning, we extended the goal-plan tree shown in Fig. 2 as follows. We introduce an additional subgoal LUGGAGEGOAL for booking a holiday which is the subgoal of purchasing a piece of luggage, such as a backpack, a travel bag or a suitcase. This subgoal LUGGAGEGOAL can be seen in Fig. 4.

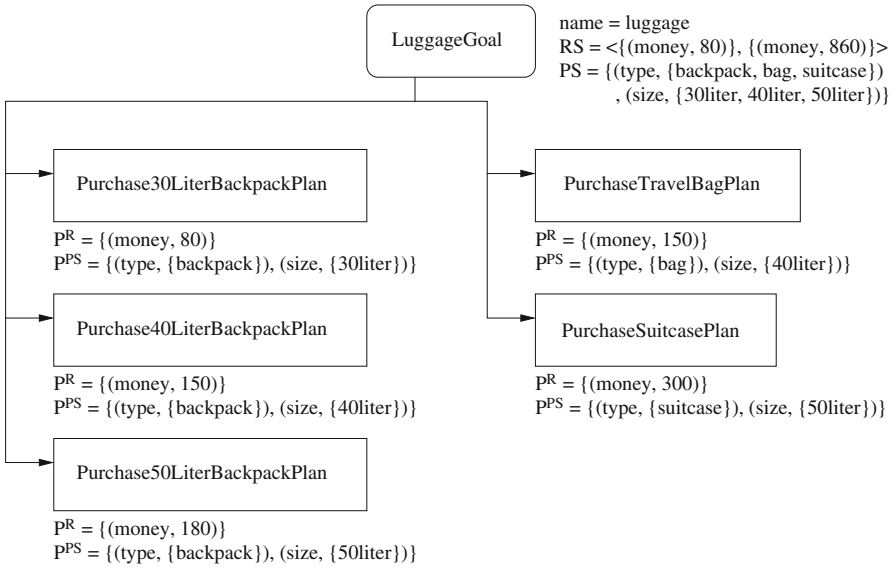


**Fig. 4** The additional implemented subgoal LUGGAGEGOAL of HOLIDAYPLAN

In this extended version of Fig. 2, the nodes HOLIDAYPLAN and HOLIDAYGOAL additionally have the properties *luggage.type* and *luggage.size* in each of their property summaries. We also introduce the following additional subgoals:

- A subgoal PAYBACKPACKERGOAL of the plan BACKPACKERPLAN with two plans to pay either by credit card or debit card (i.e., the property *payment* with respectively the values $credit$ and $debit$). We have associated with each plan the same amount of resources, $money = 100$, and BACKPACKERPLAN no longer has resources associated with it.
- A subgoal BOOKAIRPORTTRANSPORTGOAL of the plan FLIGHTPLAN which handles the goal of booking transport to get to the airport. Two plans are available, TOAIRPORTBYTAXIPLAN and TOAIRPORTBYPUBLICTRANSPORTPLAN which respectively book a taxi and book a ticket to arrive at the airport by public transport. Each of these plans has the property $payment = \{credit, debit\}$ and the monetary cost associated with these plans is respectively 50 and 20. We express the means of transport with the property $secondary\_transport$ which can be $taxi$ or $public\_transport$ and the transport towards the holiday destination (i.e., $plane$ or $train$) is now referred to by the property $primary\_transport$.
- Lastly, a subgoal PAYTRAINGOAL of the plan TRAINPLAN with two plans that distinguish between paying with credit card and debit card. The monetary cost is 115 for both plans and each of the plans has a property called *payment* with either the value $credit$ or $debit$.

Furthermore, the values of in each of their resource summaries are accordingly adjusted. We are unable to legibly display the fully annotated goal-plan tree with the above extensions, however the exact values are not important for the following purpose. We use these extensions to describe the case studies below, highlighting some of the features of our reasoning algorithms. In order to bring about certain features we also force some plans to fail. These case studies are described below.

**Table 1** Example scenario with no preferences

| AccommodationGoal | TransportGoal | LuggageGoal |
|---|---|---|
| 3* | Jetstar, taxi, credit | 30LiterBackpack |
| 5* | Jetstar, taxi, debit | 40LiterBackpack |
| Backpacker, credit | Jetstar, public transport, credit | 50LiterBackpack |
| Backpacker, debit | Jetstar, public transport, debit | TravelBag |
| | Qantas, taxi, credit | Suitcase |
| | Qantas, taxi, debit | |
| | Qantas, public transport, credit | |
| | Qantas, public transport, debit | |
| | Train, credit | |
| | Train, debit | |

### 5.2 Case study 1: no preferences

In order to provide a baseline for later results, we first consider the above scenario in which no preferences are specified. For the goal-plan tree of Fig. 2, this means that there are 9 possible outcomes. There are three choices of accommodation, and hence three possible ways to achieve the AccommodationGoal (3*, 5*, backpacker). There are also three ways to achieve the TransportGoal (Jetstar, Qantas, train), thus making $3 \times 3 = 9$ total combinations. In the extended scenario, there are four options for accommodation (the previous three, with the backpacker case splitting into two possibilities, one for payment by credit and one for payment by debit), and ten options for transport (for air transport, two airlines, two means to get to the airport and two means of payment, plus the train, split into two possibilities by means of payment). There are also five options for the new LuggageGoal. This makes a total of $4 \times 10 \times 5 = 200$ possible outcomes overall.

The outcomes for each of the three goals are given in Table 1. The 200 possibilities can be constructed by selecting one outcome from each column.

If no preferences are specified, the system will (correctly) make an arbitrary choice for one of these 200 possibilities. It may also occur that some of these may fail, such as there being no 5* accommodation available, or certain kinds of luggage not being able to be purchased. As long as there is at least one solution to each of the three goals AccommodationGoal, TransportGoal and LuggageGoal, this will not have much effect, as all it does is narrow down an arbitrary choice to a smaller number of possibilities.

Even for this relatively simple example, it should be clear that the ability to specify preferences is significant. Whilst in principle any one of these 200 possibilities is a solution for the traveller, in practice it is likely that some of these will be more suitable than others. As we shall see, we can use our preference reasoning methods to constrain the solutions provided to ones that are far more likely to satisfy the user.

### 5.3 Case study 2: the backpacker

Our first example with preferences involves those of a backpacker which are characterized by a low travel budget.

$f_1$ "I prefer to stay in backpacker accommodation rather than a hotel."

| | |
|---|---|
| | $accommodation.type = backpacker$ (0) $\gg$ $accommodation.type = hotel$ (100) |
| $f_2$ | "I prefer to travel by train rather than by aircraft." $transport.primary\_transport = train$ (0) $\gg$ $transport.primary\_transport = plane$ (100) |
| $f_3$ | "I prefer to minimize my expenses." $minimize(money)$ (0) |
| $f_4$ and $f_5$ | "If I have to travel by aircraft then I prefer to fly with Jetstar. Furthermore, in this case, I prefer to travel to the airport by public transport rather than by taxi" $transport.primary\_transport = plane : transport.book\_flight.airline = jetstar$ (0) $transport.primary\_transport = plane : transport.book\_airport\_transport.secondary\_transport = public\_transport$ (0) |
| $f_6$ and $f_7$ | "I prefer a backpack as luggage that is as large as possible." $luggage.type = backpack$ (0) $luggage.size = 50litre$ (0) $\gg$ $luggage.size = 40litre$ (50) $\gg$ $luggage.size = 30litre$ (100) |

Table 2 shows two different executions of the system. If we consider the first execution (Execution 1 in the table), observe that the agent, as expected, selects the plan for booking a backpacker accommodation (due to $f_1$) and the plan for purchasing the the largest backpack as luggage (due to $f_6$ and $f_7$). The user has not specified a preference for the method of payment hence the credit card is chosen at random.

Execution 2 is also possible as the decision between choosing a $train$ and $plane$ is neutral. This is because preference formula $f_2$ specifies a preference of $train$ over $plane$ yet preference formula $f_3$ specifies a preference to minimize the amount of resources spent. The plan FLIGHTPLAN requires at least '$money = 110$' and TRAINPLAN requires at least '$money = 115$'. When the preference formulas are evaluated for FLIGHTPLAN and TRAINPLAN we get the results shown in Table 3.

Preference formula $f_2$ is evaluated with a lower value for TRAINPLAN and preference formula $f_3$ is evaluated with a lower value for FLIGHTPLAN. Both plans receive a score of 400 which means both plans are considered equal in terms of how well they satisfy the preferences. Note that $f_4$ and $f_5$ evaluate to $v_{min}$ as the condition is false.

**Agent execution with plan failure**

In the following execution we force the failure of the plans BACKPACKERPLAN and each of the plans to purchase a backpack (i.e., PURCHASEXXLITERBACKPACKPLAN). This leads to the two executions shown in Table 4.

We will discuss the following two observations: (1) the 3* hotel was chosen as alternative to the preferred backpacker accommodation ($f_1$) because it is the cheapest alternative ($f_3$) and (2) the cheaper bag with a capacity of 40 L is chosen ($f_3$) rather than the expensive suitcase which would satisfy the preference of having a 50 L piece of luggage ($f_7$).

With regard to the first observation, we note that in both executions the agent attempts to book a backpacker hostel for accommodation but this fails. In order to achieve ACCOMMODATIONGOAL, the agent selects HOTELPLAN for execution as this is the only alternative. For its subgoal BOOKHOTELGOAL, the agent has to choose between BOOKTHREESTARHOTELPLAN and BOOKFIVESTARHOTELPLAN. Preference formula $f_3$ is now the only user preference that

**Table 2** Backpacker example without plan failure

| **Execution 1** | **Execution 2** |
|---|---|
| BackpackerCreditPlan (S)<br>BackpackerPlan (S)<br>Purchase50LiterBackpackPlan (S)<br>PayTrainDebitPlan (S)<br>TrainPlan (S)<br>HolidayPlan (S) | BackpackerCreditPlan (S)<br>BackpackerPlan (S)<br>Purchase50LiterBackpackPlan (S)<br>FlyWithJetstarPlan (S)<br>ToAirportByPublicTransportPlan (S)<br>FlightPlan (S)<br>HolidayPlan (S) |
| **Successful goals:** { pay_backpacker, accommodation, luggage, pay_train, transport, holiday }<br>**Unsuccessful goals:** { }<br>**Resources:** {(money, 395)}<br>**Properties:**<br>accommodation.pay_backpacker.payment = {credit}<br>accommodation.type = {backpacker}<br>accommodation.quality = {basic}<br>luggage.type = {backpack}<br>luggage.size = {50litre}<br>transport.pay_train.payment = {debit}<br>transport.primary_transport = {train} | **Successful goals:** {pay_backpacker, accommodation, luggage, book_flight, book_airport_transport,transport, holiday }<br>**Unsuccessful goals:** { }<br>**Resources:** {(money, 390)}<br>**Properties:**<br>accommodation.pay_backpacker.payment = {credit}<br>accommodation.type = {backpacker}<br>accommodation.quality = {basic}<br>luggage.type = {backpack}<br>luggage.size = {50litre}<br>transport.book_flight.payment = {credit}<br>transport.book_flight.airline = {jetstar}<br>transport.book_airport_transport.secondary_transport = {public_transport}<br>transport.book_airport_transport.payment = {credit}<br>transport.primary_transport = {plane} |

*S* plan succeeded, *F* plan failed

**Table 3** FLIGHTPLAN and TRAINPLAN preference evaluation

| Preference formula | FlightPlan | TrainPlan |
|---|---|---|
| $f_1$ | 100 | 100 |
| $f_2$ | 100 | 0 |
| $f_3$ | 0 | 100 |
| $f_4$ | 0 | 0 |
| $f_5$ | 0 | 0 |
| $f_6$ | 100 | 100 |
| $f_7$ | 100 | 100 |
| Total | 400 | 400 |

can guide the agent. For this reason, the agent selects the 3* hotel because it is cheaper than the the 5* hotel.

The second observation is good example of how changing circumstances can change the decisions that the agent makes. The agent needs to decide upon a piece of luggage and the relevant preferences for this decision are preferences formulas $f_3$, $f_6$, and $f_7$, which

**Table 4** Backpacker example with plan failure

| **Execution 1** | **Execution 2** |
|---|---|
| PURCHASE50LITERBACKPACKPLAN (F)<br>PURCHASE30LITERBACKPACKPLAN (F)<br>PURCHASE40LITERBACKPACKPLAN (F)<br>PURCHASETRAVELBAGPLAN (S)<br>PAYTRAINCREDITPLAN (S)<br>TRAINPLAN (S)<br>BACKPACKERDEBITPLAN (S)<br>BACKPACKERPLAN (F)<br>BOOKTHREESTARHOTELPLAN (S)<br>HOTELPLAN (S)<br>HOLIDAYPLAN (S) | PURCHASE30LITERBACKPACKPLAN (F)<br>PURCHASE40LITERBACKPACKPLAN (F)<br>PURCHASE50LITERBACKPACKPLAN (F)<br>PURCHASETRAVELBAGPLAN (S)<br>TOAIRPORTBYPUBLICTRANSPORTPLAN (S)<br>FLYWITHJETSTARPLAN (S)<br>FLIGHTPLAN (S)<br>BACKPACKERCREDITPLAN (S)<br>BACKPACKERPLAN (F)<br>BOOKTHREESTARHOTELPLAN (S)<br>HOTELPLAN (S)<br>HOLIDAYPLAN (S) |
| **Successful goals:** { luggage, pay_train, transport, pay_backpacker, book_hotel, accommodation, holiday }<br>**Unsuccessful goals:** { }<br>**Resources:** {(money, 975)}<br>**Properties:**<br>luggage.type = {bag},<br>luggage.size = {40litre},<br>transport.pay_train.payment = {credit},<br>transport.primary_transport = {train},<br>accommodation.pay_backpacker.payment = {debit},<br>accommodation.book_hotel.payment = {credit},<br>accommodation.book_hotel.quality = {3*},<br>accommodation.type = {hotel},<br>accommodation.quality = {3*} | **Successful goals:** { pay_backpacker, book_hotel, accommodation, luggage, book_airport_transport, book_flight, transport, holiday }<br>**Unsuccessful goals:** { }<br>**Resources:** {(money, 970)}<br>**Properties:**<br>accommodation.pay_backpacker.payment = {credit}<br>accommodation.book_hotel.payment = {credit}<br>accommodation.book_hotel.quality = {3*}<br>accommodation.type = {hotel}<br>accommodation.quality = {3*}<br>luggage.type = {bag}<br>luggage.size = {40litre}<br>transport.book_flight.payment = {credit}<br>transport.book_flight.airline = {jetstar}<br>transport.primary_transport = {plane}<br>transport.book_airport_transport.payment = {debit}<br>transport.book_airport_transport.secondary_transport = {public_transport} |

*S* plan succeeded, *F* plan failed

respectively deal with the cost, the type and the size of the luggage. Prior to executing any of the plans for the goal LUGGAGEGOAL, the evaluation of preferences is as shown in Table 5 (for convenience we have included the cost and size of each piece of luggage).

We omit the other preference formulas in this overview as they do not influence the decisions that are made. We can see that the backpacks of 30 and 50 L are considered equal in terms of preference satisfaction: the first one is cheaper ($f_3$) whereas the second satisfies the preference of having luggage with a capacity of 50 L ($f_7$). The agent can therefore select either PURCHASE30LITERBACKPACKPLAN or PURCHASE50LITERBACKPACKPLAN as first plan to achieve LUGGAGEGOAL and both of these situations occur in the two executions

**Table 5** Preference evaluation prior to execution

| Preference formula | Backpack 30 L (money, 80) | Backpack 40 L (money, 150) | Backpack 50 L (money, 180) | Bag 40 L (money, 150) | Suitcase 50 L (money, 300) |
|---|---|---|---|---|---|
| $f_3$ | 0 | 100 | 100 | 100 | 100 |
| $f_6$ | 0 | 0 | 0 | 100 | 100 |
| $f_7$ | 100 | 50 | 0 | 50 | 0 |
| Total | 100 | 150 | 100 | 250 | 200 |

**Table 6** Preference evaluation after failure of Purchase30LiterBackpackPlan

| Preference formula | Backpack 30 L (money, 80) | Backpack 40 L (money, 150) | Backpack 50L (money, 180) | Bag 40 L (money, 150) | Suitcase 50 L (money, 300) |
|---|---|---|---|---|---|
| $f_3$ | 100 | 0 | 100 | 0 | 100 |
| $f_6$ | 0 | 0 | 0 | 100 | 100 |
| $f_7$ | 100 | 50 | 0 | 50 | 0 |
| Total | 200 | 50 | 100 | 150 | 200 |
| Status | Failure | | | | |

**Table 7** Preference evaluation after failure of Purchase40LiterBackpackPlan

| Preference formula | Backpack 30 L (money, 80) | Backpack 40 L (money, 150) | Backpack 50 L (money, 180) | Bag 40 L (money, 150) | Suitcase 50 L (money, 300) |
|---|---|---|---|---|---|
| $f_3$ | 100 | 100 | 100 | 0 | 100 |
| $f_6$ | 0 | 0 | 0 | 100 | 100 |
| $f_7$ | 100 | 50 | 0 | 50 | 0 |
| Total | 200 | 150 | 100 | 150 | 200 |
| Status | Failure | Failure | | | |

above. The execution of either of these plans fails. We will follow Execution 2 and consider the situation after the failure of PURCHASE30LITERBACKPACKPLAN which results in the preference evaluations shown in Table 6.

The cheapest plan in the previous situation is now no longer available which means preference formula $f_3$ needs to be evaluated using the other four plans. There are now two alternatives with the same monetary cost: the 40 L backpack and the 40 L bag. These two alternatives are equal with regard to preference formulas $f_3$ and $f_7$ but they differ with regard to preference formula $f_6$. In this situation the agent can only select the 40 L backpack as this is the preferred type of luggage ($f_6$). This plan also fails which results in the situation shown in Table 7.

Given that the first two plans have failed, the agent can only select a plan out of the latter three plans: the 50 L backpack, the 40 L bag and the 50 L suitcase. The agent selects the

**Table 8** Preference evaluation after failure of Purchase50LiterBackpackPlan

| Preference formula | Backpack 30 L (money, 80) | Backpack 40 L (money, 150) | Backpack 50 L (money, 180) | Bag 40 L (money, 150) | Suitcase 50L (money, 300) |
|---|---|---|---|---|---|
| $f_3$ | 100 | 100 | 100 | 0 | 100 |
| $f_6$ | 0 | 0 | 0 | 100 | 100 |
| $f_7$ | 100 | 50 | 0 | 50 | 0 |
| Total | 200 | 150 | 100 | 150 | 200 |
| Status | Failure | Failure | Failure | | |

plan for the 50 L backpack and after this plan has failed we arrive at the situation shown in Table 8.

We can see here that the agent should select the bag rather than the suitcase as the preference of minimizing the cost ($f_3$) is considered more important than the preference of obtaining a piece of luggage with a capacity of 50 L ($f_7$). Informally, if an alternative does not fully satisfy $f_3$ (i.e., evaluate to a value of 0) it receives a value of 100 but an alternative that does not fully satisfy $f_7$ receives a value of either 50 or 100 (in this case 50). This means $f_7$ can inflict a less severe punishment than $f_3$ upon an alternative if it does not fully satisfy the user preference.

### 5.4 Case study 3: the businessman

Our second example with preferences is about a wealthy businessman whose preferences are characterized by a larger travel budget and a certain standard of travel and accommodation. This example demonstrates a conjunction in a condition and evaluating a preference using a property that have been constructed using $\mathcal{N}$ as described in Sects. 3.1.2 and 4.1.2. The preferences are:

$f_1$ "I prefer to travel by plane rather than by train."
$transport.primary\_transport = plane$ (0) $\gg$
$transport.primary\_transport = train$ (100)

$f_2$ "I prefer to fly with Qantas rather than with Jetstar."
$transport.book\_flight.airline = qantas$ (0) $\gg$
$transport.book\_flight.airline = jetstar$ (100)

$f_3$ "I prefer to stay in a 5* quality accommodation."
$accommodation.quality = 5^*$ (0)

$f_4$ "If I'm staying in a 3* accommodation then I prefer to have 40-L sized luggage."
$accommodation.quality = 3^* : luggage.size = 40L$ (0)

$f_5$ and $f_6$ "I prefer to go to the airport by taxi, which I prefer to pay by credit card."
$transport.book\_airport\_transport.secondary\_transport = taxi$ (0)
$transport.book\_airport\_transport.payment = credit$ (0)

$f_7$ "I prefer to have a suitcase as luggage."
$luggage.type = suitcase$ (50)

We will now discuss how the evaluation of preference formula $f_3$ takes place as this will show how a property that has been introduced for the goal AccommodationGoal using $\mathcal{N}$ ($accommodation.quality$), as described in Sect. 3.1.2, is evaluated. We will show that

**Table 9** Businessman example executions

| **Execution 1** | **Execution 2** |
|---|---|
| BookFiveStarHotelPlan (S)<br>HotelPlan (S)<br>PurchaseSuitcasePlan (S)<br>ToAirportByTaxiPlan (S)<br>FlyWithQantasPlan (S)<br>FlightPlan (S)<br>HolidayPlan (S) | FlyWithQantasPlan (S)<br>ToAirportByTaxiPlan (S)<br>FlightPlan (S)<br>BookFiveStarHotelPlan (F)<br>BookThreeStarHotelPlan (S)<br>HotelPlan (S)<br>PurchaseTravelBagPlan (S)<br>HolidayPlan (S) |
| **Successful goals:** { book_hotel, accommodation, luggage, book_airport_transport, book_flight, transport, holiday }<br>**Unsuccessful goals:** { }<br>**Resources:** {(money, 900)}<br>**Properties:**<br>accommodation.book_hotel.payment = {credit}<br>accommodation.book_hotel.quality = {5*}<br>accommodation.type = {hotel}<br>accommodation.quality = {5*}<br>luggage.type = {suitcase}<br>luggage.size = {50litre}<br>transport.book_airport_transport.secondary_transport = {taxi}<br>transport.book_airport_transport.payment = {credit}<br>transport.book_flight.payment = {credit}<br>transport.book_flight.airline = {qantas}<br>transport.primary_transport = {plane} | **Successful goals:** { book_flight, book_airport_transport, transport, book_hotel, accommodation, luggage, holiday }<br>**Unsuccessful goals:** { }<br>**Resources:** {(money, 950)}<br>**Properties:**<br>transport.book_flight.payment = {credit}<br>transport.book_flight.airline = {qantas}<br>transport.book_airport_transport.secondary_transport = {taxi}<br>transport.book_airport_transport.payment = {debit}<br>transport.primary_transport = {plane}<br>accommodation.book_hotel.payment = {credit}<br>accommodation.book_hotel.quality = {3*}<br>accommodation.type = {hotel}<br>accommodation.quality = {3*}<br>luggage.type = {bag}<br>luggage.size = {40litre} |

*S* plan succeeded, *F* plan failed

the way $f_3$ is evaluated varies depending on whether we are evaluating for AccommodationGoal or its subgoal BookHotelGoal. In both cases we should be able to identify a property in one of the plans for these goals to which $accommodation.quality$ can be matched. Table 9 shows two executions, one without any plan failure and the other with failure introduced.

Let us first consider the case where there is no plan failure. At the level of AccommodationGoal the agent should be able to detect that the plan HotelPlan eventually leads to a 5* accommodation. Similarly, at the level of BookHotelGoal the agent should notice that the plan BookFiveStarHotelPlan, leads to a 5* accommodation. This means that at AccommodationGoal, the property $accommodation.quality$ should be matched to $book\_hotel.quality$ of HotelPlan and at BookHotelGoal, the property $accommodation.quality$ should be matched to $quality$ of either of the plans of that goal.

Consider the case of AccommodationGoal with its two plans BackpackerPlan and HotelPlan that respectively have the properties $quality = \{basic\}$ and $book\_hotel.quality = \{3^*, 5^*\}$. We are evaluating a basic desire formula ($f_3$) of the form $name = value$ which

is the first type of basic desire formulas mentioned in Definition 8. We apply the second condition of the evaluation of preference formulas of this type to assign the value $v_{min}$ to HOTELPLAN and $v_{max}$ to BACKPACKERPLAN when evaluating $accommodation.quality = 5^*$. The agent therefore selects HOTELPLAN for execution.

Consider now the case of BOOKHOTELGOAL with its two plans BOOKTHREESTARHOTELPLAN and BOOKFIVESTARHOTELPLAN that respectively have the properties $quality = \{3^*\}$ and $quality = \{5^*\}$. The agent should now be able to detect that these properties correspond to $accommodation.quality$. However, we cannot simply match them based on their name alone as the property $quality$ could also appear in other parts of the goal-plan tree. For example, if FLYWITHJETSTARPLAN and FLYWITHQANTASPLAN had a property $quality$ then we would not want the agent to use this preference mistakenly to decide between those plans for BOOKFLIGHTGOAL.

Instead, we compute the goalpath of the property $quality$ of these plans for BOOKHOTELGOAL and we compare this goalpath to the goalpath of the property in $f_3$. To be precise, we apply the third condition of the evaluation in Definition 8. The goalpath of the properties of these plans is $accommodation.book\_hotel$ which means the full property name is $accommodation.book\_hotel.quality$.

Based on how the property $accommodation.quality$ is introduced for ACCOMMODATIONGOAL, we can conclude that a property $accommodation.book\_hotel.quality$ is part of the subtree rooted at ACCOMMODATIONGOAL and that it is therefore applicable when evaluating a property called $accommodation.quality$. The agent is therefore able to satisfy the preference formula $f_3$ by selecting BOOKFIVESTARHOTELPLAN rather than BOOKTHREESTARHOTELPLAN.

*Agent execution with plan failure*

In this case we force the plan BOOKFIVESTARHOTELPLAN to fail to analyze what happens with the condition $f_4$. Observe that preference formula $f_7$ concerns itself with the businessman's luggage and that is has a value of 50 associated with it rather than the value 0. Observe also that $f_4$ also refers to a characteristic of the luggage and that this preference formula has a value of 0 associated with it. This means that if the condition in $f_4$ is true then $luggage.size = 40L$ (0) is used when evaluating the preferences for the luggage goal. In turn, this means that this preference formula will be more important in a decision than preference formula $f_7$ as this formula has a higher value associated with it. This shows that a user can increase a value to decrease the importance of that preference formula.

## 6 Conclusion

In intelligent agents, preferences have been used for the selection [14,19] and elimination [15,17,18] of actions or plans. The preference language on which our work is based is also used by Fritz and McIlraith [10,11] to integrate preferences into the agent programming language DT-Golog.

The contributions of this work are (1) a method for annotating goals and plans with their properties, (2) the specification of user preferences in a formal language in those terms and (3) algorithms to utilize these preferences in plan selection for goals and ordering subgoals of a plan. A key characteristic of our approach is that preferences are specified independently of the goal-plan tree which can therefore be used by multiple users, each with their own preferences.

We have implemented our preference system in the agent platform Jadex [21] and have tested it on a number of examples as discussed above.

### 6.1 Discussion

A point of consideration is whether our preference language captures the 'true' preferences of the user or whether they are a rather technical and low-level representation of the user's actual preferences. For example, in our example of booking a holiday, a user could simply prefer to stay in the most luxurious accommodation and to travel with the most luxurious means of transportation. The only reason for a user to 'prefer' a less luxurious option is that the user has a sense of the associated costs with such high class options. An average student would have a smaller budget than a rich businessman yet both can specify to prefer to have the best of all worlds (i.e., accommodation, transport, luggage, etc.). The budget would then be the limiting factor which influences what the user will actually get for each of these subgoals. We feel that this is a valid point and an option worth exploring is the development of a more high-level preference language that captures these preferences. These preferences could then be translated into formulas of our preference language of which we have shown that the agent can utilize them at run time.

In our approach we aim to improve the decisions that the agent makes at run time by incorporating additional information, i.e., the user's preferences. It can be argued that that our approach merely places, prior to actual execution, some constraints on the suitable plans for a goal or the execution order of subgoals. Furthermore, one could argue that this is contrary to the underlying philosophy of BDI agents which are characterized as flexible and adaptable in a dynamical environment. Does an agent augmented with our contributions still retain these characteristics or has the agent been reduced to a complex system that essentially performs little more than "if this then do that" operations?

Our answer is that the user preferences do not limit the agent's options. Our algorithms that utilize the user preferences do not forcefully constrain or eliminate certain options even when these options are more desirable than the most preferred option according to the user preferences. As noted in our discussion of limitations, we do not take features of modern agent systems into account, such as context conditions, but their presence can still override the choice that would be made if the agent relied only on the preferences.

For example, for a goal with plans $P_1$, $P_2$, and $P_3$ with $P_1$ as most preferred option according to the user's preferences, the agent can still select the plans $P_2$ or $P_3$ if plan $P_1$ cannot be selected for execution due to the current circumstances (i.e., its context condition may not be met which prevents it from being executed).

Our preference algorithms compute a numerical value for each plan of a goal and they order the subgoals of a plan, both with regard to the user preferences. In our analysis of the presented work we focus on the choice that the agent would make using these preferences but the 'preference reasoning component' of the agent should be integrated into the agent system as a whole. Preferences may therefore guide the agent in its decisions but they cannot harm the agent's flexibility and adaptability since they are, after all, only preferences.

While it is true that the preferences can be regarded as a set of constraints that are set in stone prior to execution, the agent's decisions are still made at run time and the preferred choices do not necessarily need to be chosen at run time. Furthermore, as we have argued earlier in this section, our current preference language can be considered a rather low-level language outlines the constraints under which the agent should be executed. It may well be valid to state that a greater degree of flexibility and adaptability can be attained when a more

high-level preference language and semantics are used (i.e, a high-level preference could be translated into a variety of run time statements, depending on the circumstances at run time).

### 6.2 Limitations and future work

There are many directions for future work to further this area of research. Firstly, we have not incorporated consistency checks to see if the user has specified preferences that conflict in some or all executions of the agent system. For example, if the user prefers $backpack$ over $bag$ and $bag$ over $suitcase$ then it would be conflicting to also prefer $suitcase$ over $backpack$. More complex examples can be constructed using conditional preferences, such as preferring (1) to have a backpack as luggage, (2) to stay in a hotel but also that (3) if a backpack is purchased, a backpacker hostel is preferred for accommodation, which conflicts with (2). Preference formulas can also lead to cycles, e.g., book the hotel only if the flight is booked and book the flight only if a hotel is booked. Similarly, preferences over resource usage can also conflict. It would not be rational to specify a preference to minimize the costs and to specify that an excessive amount should be spent on each subgoal when this is clearly not the cheapest choice. It is also possible that preferences only conflict in situations that arise after a particular sequence of plan failures but not in other situations.

Secondly, we have not explored interactions that may exist between preferences. Satisfying a preference in one part of the goal-plan tree might make it impossible to satisfy preferences in other parts of the goal-plan tree. For example, a preference to book the most expensive type of accommodation, which might be satisfiable early in the agent's execution, may make it impossible to satisfy the preferences for transport and luggage which are executed later due to a lack of money. Rather than just using user preferences for the decisions at hand the agent should have a sense of how this affects decisions that inevitably need to be made later. To solve this problem a reasoning component could perhaps be developed that analyses the goal-plan tree and the preferences to determine the interactions that could arise within the agent system.

Thirdly, we have focussed on consumable resources and we have not explored the specification of and reasoning about preferences over reusable resources. Reusable resources differ from consumable resources in that they can be reused again and this can lead to different preferences. An example of a reusable resource is a communication channel, given by Thangarajah et al. [29], and the user could express preferences such as "I prefer to use at most three communication channels at the same time" and "I prefer that each communication channel is used at most five times". Future work in this direction would lead to new preference formulas for expressing these user preferences and these will need to be incorporated into the agent's reasoning algorithms.

Lastly, the assumptions that we make in our propagation rules and the way the nodes in the goal-plan tree have been annotated influence the usefulness of the computed properties. For example in Fig. 2, if BOOKFIVESTARHOTELPLAN was also annotated with $payment = \{credit\}$, we end up with $payment = \{credit\}$ and $book\_hotel.payment = \{credit, null\}$ for HOTELPLAN which is redundant. The underlying issue here is that it is not clear whether a property $prop$ of a plan and a property $prop$ of a subgoal of that plan should be considered the same or not.

An important area for future work would be to develop a preference language at a higher level to better capture the user's preferences and a mechanical way of translating those preferences into our current preference language. An important contribution of our work is applying the notion of summary information to the problem of utilizing preferences in an

agent system. We acknowledge that the preference language that is used is also very important and that improvements can be made in this area.

## References


1. Baier, J.A., & McIlraith, S.A. (2007). On domain-independent heuristics for planning with qualitative preferences. In *7th Workshop on Nonmonotonic Reasoning, Action and Change (NRAC)*.
2. Bienvenu, M., Fritz, C., & McIlraith, S.A. (2006). Planning with qualitative temporal preferences. In *KR* (pp. 134–144). AAAI Press.
3. Boutilier, C., Reiter, R., Soutchanski, M., & Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI* (pp. 355–362). AAAI Press/The MIT Press.
4. Busetta, P., Ronnquist, R., Hodgson, A., & Lucas, A. (1999). Jack intelligent agents—components for intelligent agents in java.
5. Clement, B.J., & Durfee, E.H. (1999). Theory for coordinating concurrent hierarchical planning agents using summary information. In *AAAI* (pp. 495–502).
6. Clement, B.J., & Durfee, E.H. (1999). Top-down search for coordinating the hierarchical plans of multiple agents. In *Agents* (pp. 252–259).
7. Dasgupta, A., & Ghose, A. K. (2011). Bdi agents with objectives and preferences. In A. Omicini, S. Sardina, & W. Vasconcelos (Eds.), *Declarative agent languages and technologies VIII. Lecture notes in computer science* (Vol. 6619, pp. 22–39). Berlin: Springer.
8. Fikes, R., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3/4), 189–208.
9. Fox, M., & Long, D. (2003). Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, *20*, 61–124.
10. Fritz, C., & McIlraith, S. (2005). Compiling qualitative preferences into decision-theoretic golog programs. In *Proceedings of The 6th Workshop on Nonmonotonic Reasoning, Action, and Change*. Edinburgh, UK.
11. Fritz, C., & McIlraith, S. (2006). Decision-theoretic golog with qualitative preferences. In *Proceedings of KR'2006* (pp. 153–163). Lake District, UK.
12. Gerevini, A., & Long, D. (2005). Plan constraints and preferences in pddl3—The language of the fifth international planning competition. Technical report.
13. Ghallab, M., Isi, C.K., Penberthy, S., Smith, D.E., Sun, Y., & Weld, D. (1998). Pddl—The planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
14. Hindriks, K. V., Jonker, C. M., & Pasman, W. (2008). Exploring heuristic action selection in agent programming. *ProMAS. Lecture notes in computer science* (Vol. 5442, pp. 24–39). Berlin: Springer.
15. Hindriks, K. V., & Birna van Riemsdijk, M. (2008). Using temporal logic to integrate goals and qualitative preferences into agent programming. *DALT. Lecture notes in computer science* (Vol. 5397, pp. 215–232). Berlin: Springer.
16. Ingrand, F. F., Georgeff, M. P., & Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, *7*(6), 34–44.
17. Myers, K.L., & Morley, D.N. (2001). Human directability of agents. In *K-CAP* (pp. 108–115). ACM.
18. Myers, K.L., & Morley, D.N. (2002). Resolving conflicts in agent guidance. In *Proceedings of the AAAI-02 Workshop on Preferences in AI and CP: Symbolic Approaches*.
19. Nguyen, A., & Wobcke, W. (2006). An adaptive plan-based dialogue agent: Integrating learning into a bdi architecture. In *AAMAS* (pp. 786–788). ACM.
20. Padgham, L., & Singh, D. (2013). Situational preferences for bdi plans. In M.L. Gini, O. Shehory, T. Ito, & C.M. Jonker (Eds) *AAMAS* (pp. 1013–1020). IFAAMAS.
21. Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. *Multi-agent programming* (Vol. 9, pp. 149–174). New York: Springer.
22. Rao, A.S., & Georgeff, M.P. (1992). An abstract architecture for rational agents. In *KR* (pp. 439–449).
23. Rnnquist, R. (2007). The goal oriented teams (gorite) framework. In M. Dastani, A. El Fallah-Seghrouchni, A. Ricci, & M. Winikoff (Eds.), *PROMAS. Lecture notes in computer science* (Vol. 4908, pp. 27–41). Berlin: Springer.
24. Shaw, P. H., & Bordini, R. H. (2007). Towards alternative approaches to reasoning about goals. In M. Baldoni, T. Cao Son, M. Birna van Riemsdijk, & M. Winikoff (Eds.), *DALT. Lecture notes in computer science* (Vol. 4897, pp. 104–121). Berlin: Springer.

25. Shaw, P. H., & Bordini, R. H. (2010). An alternative approach for reasoning about the goal-plan tree problem. In H. Coelho, R. Studer, & M. Wooldridge (Eds.), *ECAI. Frontiers in artificial intelligence and applications* (Vol. 215, pp. 1035–1036). Amsterdam: IOS Press.
26. Shaw, P.H., Farwer, B., & Bordini, R.H. (2008). Theoretical and experimental results on the goal-plan tree problem. In L. Padgham, D.C. Parkes, J.P. Müller, & S. Parsons (Eds), *AAMAS (3)* (pp. 1379–1382). IFAAMAS.
27. Thangarajah, J., Padgham, L., & Winikoff, M. (2003). Detecting & exploiting positive goal interaction in intelligent agents. In *AAMAS* (pp. 401–408). ACM.
28. Thangarajah, J., Sardiña, S., & Padgham, L. (2012). Measuring plan coverage and overlap for agent reasoning. In W. van der Hoek, L. Padgham, V. Conitzer, & M. Winikoff (Eds), *AAMAS* (pp. 1049–1056). IFAAMAS.
29. Thangarajah, J., Winikoff, M., Padgham, L., & Fischer, K. (2002). Avoiding resource conflicts in intelligent agents. In *ECAI* (pp. 18–22). IOS Press.
30. Toranzo, C., Errecalde, M., & Ferretti, E. (2014). A framework for multi-criteria argumentation-based decision making within a bdi agent. *Journal of Computer Science and Technology*, *14*(1), 46–54.
31. Winikoff, M., & Cranefield, S. On the testability of bdi agent systems. Discussion Paper 2008/03, Department of Information Science, University of Otago, http://eprints.otago.ac.nz/793/. Accessed 15 Jan 2015.