No Pizza for You: Value-based Plan Selection in BDI Agents

Stephen Cranefield
University of Otago
stephen.cranefield
@otago.ac.nz

Michael Winikoff
University of Otago
michael.winikoff
@otago.ac.nz

Virginia Dignum
TU Delft
M.V.Dignum@tudelft.nl

Frank Dignum
Utrecht University
F.P.M.Dignum@uu.nl

Abstract

Autonomous agents are increasingly required to be able to make moral decisions. In these situations, the agent should be able to reason about the ethical bases of the decision and explain its decision in terms of the moral values involved. This is of special importance when the agent is interacting with a user and should understand the value priorities of the user in order to provide adequate support. This paper presents a model of agent behavior that takes into account user preferences and moral values.

1 Introduction

Social assistive software and robots is an increasing area of research and development [Oishi *et al.*, 2010]. Such systems are envisioned as supporting their users in daily activities, such as medication monitoring and agenda reminders. The use of such systems, often involving vulnerable users, raises substantial ethical concerns: How is users' privacy protected? Which tasks should the system be allowed to perform and who can regulate and monitor this? Will artificial caregivers displace human caregivers, negatively affecting both client welfare and health-care provider jobs? How to ensure that the system is aligned with and able to uphold the moral, societal and legal values of the user and society? This paper focuses on this last question and proposes means to integrate values into the planning of agent activities.

Ethical decision making can be understood as action selection under conditions where principles, values, and social norms play a central role in determining which behavioral attitudes and responses are acceptable. However, the way agents choose between different possible courses of action ("plans"), is often left to the programmer of the agent. This may be done by statically prioritizing the plans by ordering them in a file or by using (implicit) criteria that are predetermined (and usually are utility-, resource- or time-optimizing).

Although this usually works well in applications where agents only have goals related to a particular type of situation, it does not transfer to applications where agents have several different tasks that are not directly related, e.g. an elderly companion robot or an e-health coach. In these applications different interactions might require different criteria to optimize and long-term criteria might differ from short-term objectives. Thus a person can eat a cake on his birthday while trying to lose weight over a longer period. At first sight it may seem inconsistent, but there is a balance between enjoyment of a birthday and long-term health. However, it would be better if the person would get the cake by walking to the shop rather than going there by car (provided it is within walking distance). In other words, the health criterion not only plays a role in the eating decision, but also in the transport decision.

In this paper we present a *computational mechanism* for using values to select between (hierarchical) plans in a consistent manner. Using values has two advantages. First there exists an extensive literature on human values and their relations. It shows that people have a common base system of values where the difference between people lies in the priorities they give to the values. It also indicates that values are relatively stable over the life span of a person. Thus they can be used as an underlying stable mechanism for decision making. Note that we do not claim that every decision is explicitly based on some value. Many decisions are made based on norms and longer term goals. However these norms and goals are often chosen based on the value system. Thus these decisions are indirectly influenced by the values.

A second advantage of using values is the ability to explain decisions over different situations in a relatively simple manner. This is important to generate a level of trust in the system by a human user in that the user can maintain a model of the system and can predict its future actions based on that model. Having underlying values explaining a wide variety of decisions in different situations makes this much easier than having explicit rules for all possible situations not directly related.

In the rest of the paper we will first sketch some background literature, indicating the added value of our approach (Section 2). Section 3 then presents a scenario to illustrate the use of values in the deliberation of a companion agent that advises a person on healthy living. Section 4 describes how this scenario can be modeled in our framework, and Section 5 defines the computational mechanism for reasoning about values. Section 6 concludes the paper and outlines future work.

2 Literature

This paper proposes a novel approach to plan selection in reactive planning, in which societal, moral and legal values of users guide the planning process. As such it is based on current work on value-sensitive design and on planning. Values (e.g. honesty, beauty, respect, environmental care, Self-Enhancement) are key drivers in human decision making (see e.g. [Rokeach, 1973; Schwartz, 2012]). As such, values can be seen as criteria to measure the difference between two situations or for comparing *alternative* plans. Values are abstract and context dependent, and therefore cannot easily be measured directly. For example, the value *wealth* can include assets other than money, but can be approximated by the amount of money someone owns. Miceli and Castelfranchi [1989] discuss in depth the consequences of this indirect use of values.

Values combine two core properties: (1) *Genericity:* values are generic and can be instantiated in a wide range of concrete situations, and therefore can be seen as a very abstract goal (e.g. eating well, exercising and avoiding stress all contribute to the value 'health'). (2) *Comparison:* values allow comparison of different situations with respect to that value (e.g. according to the value 'health', salad is preferred over pizza). In this sense, values become metrics that measure the effects of actions in different dimensions.

For strengthening their decisions and covering a wider range of decisions, individuals tend to rely on the influence of multiple values (e.g. environmental care and wealth). However, for certain decisions, the values involved can lead to contradictory preferences. For example, cycling to work through the rain might be good for the environment, but will leave you soaked and giving a bad impression in an important meeting.

In order to handle these contradictions, value-systems *internally order* values. There are two dimensions along which this ordering takes place. First, there is an intrinsic opposition between different basic values. This intrinsic opposition is depicted by Schwartz as a circle in which the values are placed (see Figure 1, adapted from [Schwartz, 2012]). Values that are close together on the circle work in the same direction and values on opposite sides drive people in opposite directions. For example, "achievement" and "benevolence" are conflicting values. This means that generally trying to do something that is primarily good for one's own benefit (Self-Enhancement) is not necessarily the best for society. For example, making more profit by paying low wages is good for the employer's wealth but bad for the wealth of employees. However, opposition of values does not mean that one value excludes another value. It mainly means that they are in general "pulling" in different directions and a balance must be found. For example, if wages are too high the company might go bankrupt and no one profits anymore.

The second ordering is a personal preference one. Some values are given a relative *importance* over others. When evaluating a decision with conflicting values, alternatives that satisfy the most important values tend to be preferred (e.g. if health is more important than wealth then a person will buy healthy food even if it is more expensive than junk food). So, the importance of values for a person determines how the balance is struck between conflicting values.

Value-Sensitive Design (VSD) is a theoretically grounded approach to the design of technology that accounts for human

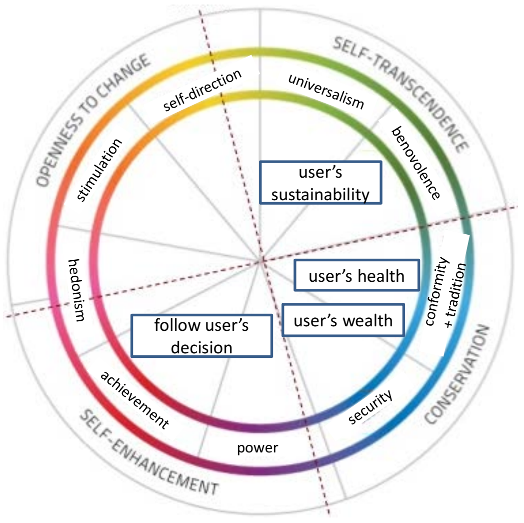


Figure 1: Schwartz's value model annotated with scenario

values in a principled and comprehensive manner [Friedman *et al.*, 2013; van den Hoven, 2007]. A crucial step in VSD is the translation of values into design requirements. In this paper, we follow the *value hierarchy* approach proposed by van de Poel [2013], which links values, norms and design requirements or goals through a *'for the sake of'* relationship. Formally, values can be linked to specific tasks or sub-plans in a plan tree, using a *'counts as'* formalization, which enables reasoning about the motives to choose for a specific course of action. This is similar to work done in normative systems to link norms to agent actions [Grossi *et al.*, 2006] and implements the properties of values as described above.

Our proposal to link values to the plans of agents is similar to how Visser *et al.* [2016] add preferences to the plans of BDI (Belief-Desire-Intention) agents [Rao and Georgeff, 1995]. However, using values gives us a way to create consistency between decisions over different actions resulting in a generic approach that is still consistent with Visser *et al.* [2016].

Also related is work on integrating planning into BDI languages [Sardiña and Padgham, 2011; Sardiña *et al.*, 2006]. This work proposes the CANPLAN language. CANPLAN exploits the similarities between BDI languages and HTN planning to provide a construct $\mathsf{Plan}(P)$ which does lookahead planning for the plan P. This work differs from our work in that they are doing full lookahead planning to find a course of action, whereas we are considering an under-constrained situation (with multiple options), and using the consequences of the available options to select from among them. Meneguzzi and de Silva [2015] survey other related work that incorporates planning into the BDI architecture.

Finally, Bordini *et al.* [2002] implemented one of AgentSpeak's selection functions by integrating quantitative reasoning using TÆMS [Decker, 1996]. A key difference between our work and theirs is that they focused on intention selection, i.e. selecting *which* intention to execute next, whereas we are dealing with selecting *how* to achieve a given sub-goal.

3 Scenario

Given the growing elderly population in many countries, the development of health-care robots and virtual assistants is a significant area of research and, at least in Japan, a serious option for elderly care. The functionalities and requirements for these Elderly Care Artificial Systems (ECAS) are very diverse, but it is certain that ECAS will need to take decisions on behalf and for their users. In this paper, we explore the situation in which a extremely simplified ECAS should order and serve a meal to its user [McColl and Nejat, 2013].

In order to decide on the most suitable meal, the robot must consider the preferences of the user, the dietary prescriptions, the financial implications, the ease and speed of delivery, possibly the carbon footprint of the choice, and other issues. Besides contextual constraints (time, money, availability) this decision is guided by moral and societal values held by the user and his/her current priorities. An ECAS's highest value is to assist its user, which should be done in accordance with the values of the user. In this meal-assistance scenario, relevant high level user values are Self-Enhancement, Conservation and Self-Transcendence, using the Schwartz classification [Schwartz, 2012]. It should be noted that the values of Self-Enhancement and Self-Transcendence pull in different directions and need thus to be balanced carefully.

According to van de Poel [2013], the abstract values are translated into concrete values to govern the ECAS's actions, e.g. 'follow user's desires' or 'ensure user's health', which are finally linked to concrete system goals, e.g. 'serve the desired meal' or 'serve a healthy meal'. Figure 2 depicts the value hierarchy, linking user values to concrete goals for the ECAS's actions. If the desired meal is neither sustainable nor healthy a choice has to be made about which value to support most, and the priorities between values determine the outcome of the choice. In our approach, plans are selected using a multi-criteria optimisation (via a weighted sum). Here each criterion measures the extent to which a particular value is currently satisfied, given the plans for its child goals.

4 Model

In modeling the problem we take two aspects into account: the agent's goals and plans; and the values and their relationships.

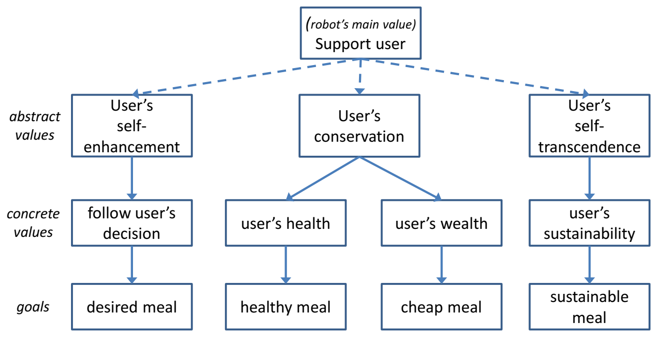


Figure 2: Value Tree

4.1 Goals and Plans

We begin with the agent's goals and plans. We assume BDI-style plans (excluding cycles), where each plan has a goal that it achieves, a context condition that indicates in which situations the plan can be used, and a plan body. Following AgentSpeak(L) and Jason [Rao, 1996; Bordini *et al.*, 2007] we consider a plan body to be a sequence of steps (actions or sub-goals), but this restriction can be easily relaxed. Space precludes a detailed exposition of BDI languages.

The options for BDI agents to achieve their goals can be represented as a goal-plan tree, where a goal node has as children the plans that can be used to achieve it (an "or" relationship), and a plan node has as children its sub-goals ("and", or more precisely "seq"). We extend our BDI language by annotating actions or plans with their effects on the Value State (defined below), similar to how Visser *et al.* [2016] extend goal-plan trees with preferences. For example, the plan evil_pizza (left side of Figure 3) is annotated "Desire: +20" indicating that it increases the Value State of Desire by 20.

Figure 3 shows a goal-plan tree for the scenario. The top-level goal has a single plan with three sub-goals: choosing a meal, preparing the meal, and consuming the meal. There are three choices: toast (which is highly unhealthy due to inadequate nutrition, but both sustainable and cheap), a frozen meal (most healthy), and pizza (most desired) with two possible providers: a local pizza company (within walking distance), and a multinational "evil" pizza company that is cheaper but less sustainable. Figure 3 elides the bindings and context conditions that are needed to constrain the plans chosen to achieve the prepare and consume sub-goals to be consistent with the selection made earlier when achieving choose. The annotations "+" and "-" before a variable's name indicate whether the variable has values produced by the goal in question ("-"), or whether the goal in question uses values produced by earlier goals ("+"). The weather goal (bottom left) instantiates a variable (W) that depends on the environment. For such 'query goals' we assume we have a probabilistic model of the possible variable bindings, which appear as child nodes of the query goal.

4.2 Values

A *Value* is an abstract representation of a human driver (e.g. Self-Enhancement, Self-Conservation, Self-Transcendence). As we have explained in Section 2, values can be in conflict and also have a relative importance. These two aspects together determine how much time or effort a person (or in our case agent) will spend to promote the different values.

We start by determining a target for each value, $T(v)$. This is a number that is compared to the current Value State to determine the current need to advance that value. To reflect observations in human behavior [Schwartz, 2012], a design constraint in our current model is that conflicting values (that pull decisions in opposite directions) cannot both be important at the same time. For example, the annotations assigned to plans and the relative importance of goals ("salience", defined below) should reflect that decisions related to Self-Enhancement can potentially be at odds with those related to Self-Transcendence, and therefore often cannot be realized

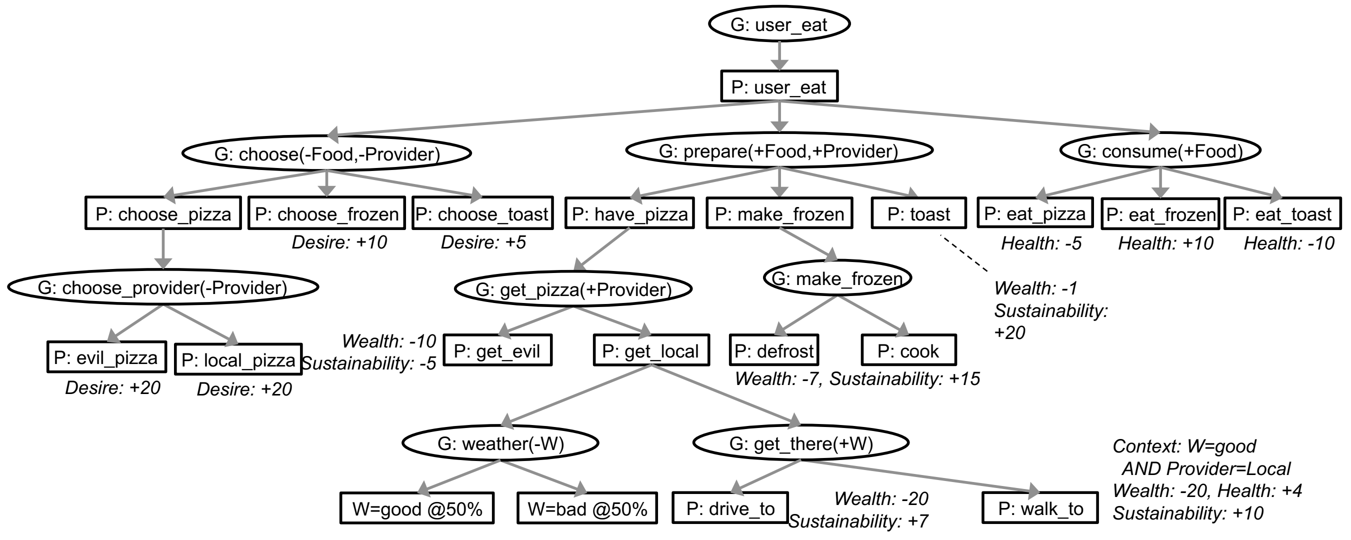


Figure 3: Goal-plan tree annotated with values

together. On the other hand, values that are very much aligned should have compatible importance. Finally, for simplicity's sake, we assume (for now) that the $T(v)$ remain constant during the life of the agent.

At each point in time, for a given value v, the agent also has a Value State, $S(v)$, that represents the current level of "satisfaction" for the value, i.e. the current level at which the value is experienced. For example, if the user has been deprived of coffee for a long time, then its hedonism Value State might be low, which will make the need to satisfy it more urgent. Formally, $S(v)$ assigns to each value type a number. The Value State is represented numerically on the same scale as $T(v)$, i.e. $S(v) < T(v)$ indicates that the Value State is below the target. We assume that $S(v)$ decays over time (the precise decay pattern needs to be specified, but our approach is agnostic). Decay represents the fact that if for some time nothing has been done to promote that value, its satisfaction will decrease. Decay is calculated at the 'concrete value' nodes of the value tree (see Figure 2) using the same decay function for all nodes. This is propagated up to the root in a very simple way: if all children of a node decay then the node itself also decays (using the same decay function).

Values also have a *current importance*, $CI(v)$, consisting of two components: the salience of the value in a situation $sal(v, g)$ and the difference between the current Value State and the target amount of that value $(T(v) - S(v))$ [Di Tosto and Dignum, 2013]. We assume the salience of a value to be given for each goal. We calculate $CI(v)$ as

$$CI(v) = sal(v, g) \times f(T(v) - S(v))$$

using some function $f(x)$ (cf. Section 5.1 for the actual implementation) to model that the importance of a value does not grow linearly with the distance of its satisfaction to its target, but can have some threshold or other non-linear shape. However, we retain generality by permitting an arbitrary user-specified function for the importance of a given value, as a function of the distance from the Value State to the target.

Priorities between values are thus not static, but can depend on the context. For example, if $sal(wealth, \textsf{prepare})$ is very high, but the $S(v)$ for wealth is well above $T(v)$, then the current priority of the value "wealth" may be quite low.

4.3 Extending AgentSpeak

We now briefly explain how the AgentSpeak language is extended to accommodate reasoning about courses of action using values at runtime. Specifically we use the Jason language [Bordini *et al.*, 2007] (which extends Rao's original AgentSpeak(L)). Although we do make use of some of Jason's extensions, our work can also be easily adapted to apply to the original AgentSpeak, and to other BDI languages.

Inspired by Sardiña and Padgham [2011] and Sardiña *et al.* [2006] we introduce a construct "$\mathcal{VBR}(G)$" denoting that sub-goal G should be achieved, but with the selection of choices to achieve it being guided by value-based reasoning.

At compile-time we pre-process this construct away[1], resulting in a collection of constraints, and a modified Jason program. Where the original program has $\mathcal{VBR}(G)$, the modified program invokes an external constraint solver to find a best course of action (in accordance with the values and their Value State), and it then uses the recommended course of action to guide the achievement of G.

We transform the agent program by firstly identifying the plans that are involved in achieving G, either directly because their trigger is $!G$, or indirectly (recursively) because their trigger is a goal that is a sub-goal of a plan that is used to achieve G. Each of these plans is then modified by: (i) adding an annotation[2], and (ii) adding an additional test to the context condition. The annotation is used to carry an additional argument capturing the *choices* gener-

[1]Doing this allows Jason to be used without modification, but it does mean that plans cannot be updated at run-time.

[2]This is a Jason construct that allows (e.g.) a goal to have additional information associated with it.

ated by the value-based reasoning[3]. The additional test in the context condition is that the choice specified by the value-based reasoning is this plan. Formally, we use the annotation $choice(Choice)$ to capture, when the plan is called, the path through the goal-plan tree generated by the value-based reasoning. We define $Choice[G]$ to be the numerical index of the plan chosen to achieve G. The additional context condition for the ith plan to achieve G is then a test that either $Choice$ is not ground, or that it indicates the current plan, i.e. $(\neg ground(Choice)) \vee Choice[G] = i$. For example, the Jason plan $+!g : c \leftarrow P$ (where $!g$ is the goal being handled, c the context condition, and P the plan body) is modified to the following (where P' is P where sub-goals have the annotation $[choice(Choice)]$ added).

$$\begin{aligned} &+!g[choice(Choice)] \\ &\quad : \quad ((not\ .ground(Choice)) \mid Choice[g] = i) \wedge c \\ &\quad \leftarrow \ P'. \end{aligned}$$

Secondly, we generate the constraints from the goal-plan tree (which is itself derived from the program). Note that the constraints do not change, and hence can be generated at compile-time. We discuss this process below in Section 5.

Finally, we replace $\mathcal{VBR}(G)$ with the sequence of steps: $.callSolver(Choice)$ and $!G[choice(Choice)]$. The first step (implemented as an internal action) invokes an external solver to solve the constraints. The second step calls the sub-goal G, but with the output from the solver provided as an annotation. An exception is made for plans with an initial query goal (see Section 5.1). These are optimised using an expected value approach. A second optimisation is needed for the subgoals of such a plan after the query has been executed to instantiate its variable and before the plan is executed.

5 Process

The process for making value-based decisions has two steps. Firstly, we take the constraint problem that has been generated from the goal-plan tree (Section 5.1) and use a standard constraint solver to solve it, yielding a best course of action for the current situation. Secondly, we implement the selected course of action (by achieving[4] the goal $!G[choice(Choice)]$) in the modified Jason program. In Section 5.2 we analyse the scenario using this implementation.

5.1 Finding a best Course of Action

In order to determine a best course of action we generate constraints from the program, and then solve the constraints. We use constraints because, unlike [Thangarajah *et al.*, 2002; Visser *et al.*, 2016], we need to deal with dependencies between different parts of the tree.

Before defining the mapping we need to formally define goal-plan trees. A goal plan tree is represented by its root node. A node N comprises a name N^n, an optional annotation N^a that is a tuple of changes to the Value State, an optional context condition N^{cc} (a logical formula), the input (N^i) and output (N^o) variables (both sets of variables), an optional binding N^b of the form $var = val$ (allowing for multiple variables, i.e. var can be a tuple of variables and val a tuple of constants), a type N^t (either g or p for "Goal" or "Plan"), and a list of N^c child nodes $N^C = \{N^C_1, \ldots, N^C_{N^c}\}$ (note that N^c, lower case "c", is the *number* of children and N^C is the set of child nodes N^C_i).

We now define the mapping from a goal-plan tree to a constraint solving problem. For each node we declare a variable with the name of the node and type of tuple of Value State changes. We also declare variables that appear in any node's N^i or N^o, and, for nodes that are goals, we declare a variable c_N^n (c for "chosen") that is an array of N^c Booleans (0 or 1) constrained so that exactly one of them is true. The variable c_N^n represents which plan is chosen to achieve the goal corresponding to node N. Formally we define $d(N)$ to denote these declarations associated with a node N.

We generate the following constraints. For a plan node the value of the node is the sum of the node's children. For a goal node we add a constraint for each child of the form $c\_N^n[i] = 1 \Rightarrow N^n = (N^C_i)^n \wedge N^{cc} \wedge N^b$, i.e. if the choice is i, then the value of the node N is the value of its ith child N^C_i, and the context condition and binding of the ith child apply. Finally, if a node has an annotation, then the value of the node is simply that annotation (and the node's type and children are ignored[5]). Formally:

$$c(N) \quad = \quad \begin{cases} N^n = N^a & \text{if } N^a \text{ is present} \\ N^n = \sum_{i=1}^{N^c} N_i^{Cn} & \text{else if } N^t = p \\ \Sigma_i c\_N^n[i] = 1 & \text{otherwise} \\ \quad \wedge \ \bigwedge_{i=1}^{N^c} c\_N^n[i] = 1 & \\ \qquad\qquad \Rightarrow \ (N^n = (N^C_i)^n \wedge N^{cc} \wedge N^b) & \end{cases}$$

Plan nodes with an initial query goal[6] are optimised using an expected value approach. Copies of the plan node's subtree with the query goal removed are made for each binding, with distinct new node names N^n_i replacing each N^n in the original subtree. Distinct renamed copies of the query goal's variable are made for each copied tree and set to the respective binding. Constraints are generated as shown above for each copy, and finally the following constraint is added: $N^n = \Sigma_i p_i N^n_i$, where the p_i are the binding probabilities.

We apply the functions $d(N)$ and $c(N)$ to all nodes and collect the results. This gives us the constraints and variables.

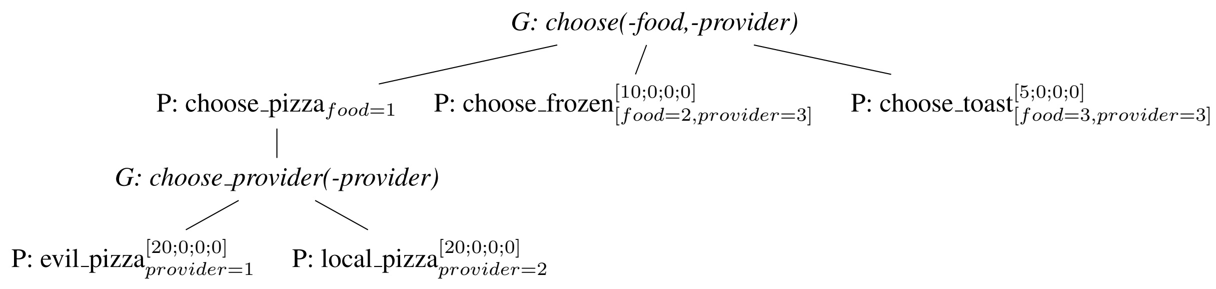
Figure 4 shows an extract of a goal-plan tree (which corresponds to a Jason program, not shown for space reasons), and the constraints that are generated (using our implementation) from the goal-plan tree. In the tree goal nodes are *italic* and a node of the form $N^{\delta}_{x=c}$ indicates that the plan labeled N is annotated with value change δ, and has a plan selection constraint or effect that constrains variable x to equal c (and $[x_1 = c_1, x_2 = c_2]$ constrains x_i to equal C_i). Here $food$ values represent pizza (1), frozen food (2), and toast (3), while

[3]If using a BDI language other than Jason, then a copy of each plan can be added, extended by an additional argument.

[4]If the goal achievement fails, then failure handling will be used, e.g. for Jason using a pattern [Bordini *et al.*, 2007, Section 8.2]. One slight wrinkle is that we need to ensure that if $!G[choice(Choice)]$ fails, that the recovery is not bound to the choices, i.e. use $!G$ in the recovery plan, not $!G[choice(Choice)]$.

[5]Therefore if a node has an annotation then the tree below it cannot contain any bindings.

[6]We do not currently handle more complex uses of query goals.


G: choose(-food,-provider)
P: choose_pizza$_{food=1}$
P: choose_frozen$^{[10;0;0;0]}_{[food=2,provider=3]}$
P: choose_toast$^{[5;0;0;0]}_{[food=3,provider=3]}$
G: choose_provider(-provider)
P: evil_pizza$^{[20;0;0;0]}_{provider=1}$
P: local_pizza$^{[20;0;0;0]}_{provider=2}$

Vars: food, provider : int; c_g_choose:$\{0,1\}^3$, c_choose_provider:$\{0,1\}^2$;
g_choose, p_choose_pizza, g_choose_provider, p_evil_pizza, p_local_pizza, p_choose_frozen, p_choose_toast : Int^4;
Constraints:
p_evil_pizza$= (20,0,0,0)$, p_local_pizza$= (20,0,0,0)$, p_choose_frozen$= (10,0,0,0)$, p_choose_toast$= (5,0,0,0)$
$\sum_{i=1}^{3}$ c_g_choose$[i] = 1 \wedge \sum_{i=1}^{2}$ c_g_choose_provider$[i] = 1$
c_g_choose$[1] = 1 \Rightarrow$ (g_choose = p_choose_pizza $\wedge$ food $= 1$)
c_g_choose$[2] = 1 \Rightarrow$ (g_choose = p_choose_frozen $\wedge$ food $= 2 \wedge$ provider $= 3$)
c_g_choose$[3] = 1 \Rightarrow$ (g_choose = p_choose_toast $\wedge$ food $= 3 \wedge$ provider $= 3$)
p_choose_pizza = g_choose_provider
c_g_choose_provider$[1] = 1 \Rightarrow$ (g_choose_provider = p_evil_pizza $\wedge$ provider $= 1$)
c_g_choose_provider$[2] = 1 \Rightarrow$ (g_choose_provider = p_local_pizza $\wedge$ provider $= 2$)

Figure 4: Example goal-plan tree and the constraints generated from it ("c_" is short for "chosen_")

$provider$ values represent evil pizza (1), local pizza (2), and home (3).

The objective function that we minimize, subject to these constraints, is $CI(v) = sal(v,g) \times f(T(v) - S(v))$, summed over the different values. We use the function $f(x) = (\max\{x, 0\})^2$ (although other functions could obviously be used). Thus, the constraints are solved while minimizing:

$$\sum_{v \in \mathcal{V}} sal(v,g) \times (\max\{T(v) - S(v)\ ,\ 0\})^2$$

where $\mathcal{V}$ is the set of all value types. The output from the constraint solver includes the values for each of the choices c_N^n - this allows us to guide the selection of plans to follow the recommended course of action.

5.2 The Scenario Revisited

The goal-plan tree in Figure 3 has been encoded and our implementation of the mapping in Section 5.1 has been used to generate constraints in MATLAB using the YALMIP (yalmip.github.io) optimisation library. We use MOSEK (mosek.com), a state-of-the-art industrial optimiser, as the underlying solver. The internal representation of the problem has 194 variables and 370 constraints.

In a situation where all four values (desire, health, wealth and sustainability) have equal salience, their targets are all 100, and their Value States are $(110, 50, 80, 20)$, the best choice (found by the constraint solver in a fraction of a second[7]), is to get and eat pizza from the local provider. The computed Value State change is $(20, -3, -20, 8.5)$, where the health and sustainability values are expected values from either walking or driving, depending on the weather.

However, in other situations, different decisions are appropriate. For instance, in a situation where all four values have equal importance (i.e. equal CI), the best choice is a frozen meal (with Value State change $(10, 10, -7, 15)$). On the other hand, if wealth is somewhat more important (i.e. $CI(wealth)$ is sufficiently greater than the other values' CI), and sustainability not important at all, then the cost saving offered by evil pizza makes it the best choice (Value State change of $(20, -5, -10, -5)$). Finally, if wealth is the overriding criterion, then toast becomes the best choice $(5, -10, -1, 20)$.

6 Conclusions

In this paper, we propose a value-based approach to plan selection, that takes into account societal and ethical values that influence decision-making. Using values as basis for deliberation supports both stability over time and the ability to explain decisions over different situations in a relatively simple and cohesive manner. We show the potential of this approach on a simple scenario of a Elderly Care Artificial System (ECAS) as an example of social assistive technology.

We have described a mechanism for BDI agents to make decisions using value-based reasoning. The mechanism uses an external constraint solver, and does not require changing the BDI language or its implementation.

Future work is needed on the evaluation of the scalability and broader applicability of the approach, on a mechanism to use values to provide explanations of an agent's behaviour, and on consideration of multi-agent decision making.

[7] 0.2324 seconds, obtained from YALMIPs *solvertime* property and averaged over 10 runs on a 2.6GHz Intel Core i7 running Windows 7.

References

[Bordini *et al.*, 2002] Rafael H. Bordini, Ana L. C. Bazzan, Rafael de Oliveira Jannone, Daniel M. Basso, Rosa Maria Vicari, and Victor R. Lesser. AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pages 1294–1302. ACM, 2002. doi:10.1145/545056.545122.

[Bordini *et al.*, 2007] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, 2007. ISBN 0470029005. doi:10.1002/9780470061848

[Decker, 1996] Keith Decker. TÆMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 16, pages 429–448. Wiley, 1996. http://mas.cs.umass.edu/paper/159.

[Di Tosto and Dignum, 2013] Gennaro Di Tosto and Frank Dignum. Simulating Social Behaviour Implementing Agents Endowed with Values and Drives. In Francesca Giardini and Frédéric Amblard, editors, *International Workshop on Multi-Agent-Based Simulation XIII (MABS): Revised Selected Papers*, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-38859-0_1.

[Friedman *et al.*, 2013] Batya Friedman, Peter H. Kahn, Alan Borning, and Alina Huldtgren. Value sensitive design and information systems. In Neelke Doorn, Daan Schuurbiers, Ibo van de Poel, and Michael E. Gorman, editors, *Early engagement and new technologies: Opening up the laboratory*, pages 55–95. Springer Netherlands, Dordrecht, 2013. doi:10.1007/978-94-007-7844-3_4.

[Grossi *et al.*, 2006] Davide Grossi, John-Jules C.H. Meyer, and Frank Dignum. Classificatory aspects of counts-as: An analysis in modal logic. *Journal of Logic and Computation*, 16(5):613–643, 2006. doi:10.1093/logcom/exl027.

[McColl and Nejat, 2013] Derek McColl and Goldie Nejat. Meal-time with a socially assistive robot and older adults at a long-term care facility. *Journal of Human-Robot Interaction*, 2(1):152–171, 2013. doi:10.5898/JHRI.2.1.McColl.

[Meneguzzi and de Silva, 2015] Felipe Meneguzzi and Lavindra de Silva. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, 30(1):1–44, 2015. doi:10.1017/S0269888913000337.

[Miceli and Castelfranchi, 1989] Maria Miceli and Christiano Castelfranchi. A Cognitive Approach to Values. *Journal for the Theory of Social Behaviour*, 19(2):169–193, 1989. doi:10.1111/j.1468-5914.1989.tb00143.x.

[Oishi *et al.*, 2010] Meeko Mitsuko K. Oishi, Ian M. Mitchell, and H.F. Machiel Van der Loos. *Design and use of assistive technology: social, technical, ethical, and economic challenges*. Springer Science & Business Media, 2010. doi:10.1007/978-1-4419-7031-2

[Rao and Georgeff, 1995] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor R. Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multiagent Systems*, pages 312–319. The MIT Press, 1995.

[Rao, 1996] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perrame, editors, *Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *LNAI*, pages 42–55. Springer, 1996.

[Rokeach, 1973] M. Rokeach. Rokeach Values Survey. In *The Nature of Human Values*. 1973.

[Sardiña and Padgham, 2011] Sebastian Sardiña and Lin Padgham. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70, 2011. doi:10.1007/s10458-010-9130-9.

[Sardiña *et al.*, 2006] Sebastian Sardiña, Lavindra de Silva, and Lin Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1001–1008. ACM, 2006. doi:10.1145/1160633.1160813.

[Schwartz, 2012] S.H. Schwartz. An Overview of the Schwartz Theory of Basic Values. *Online Readings in Psychology and Culture*, 2(1), 2012. doi:10.9707/2307-0919.1116.

[Thangarajah *et al.*, 2002] John Thangarajah, Michael Winikoff, Lin Padgham, and Klaus Fischer. Avoiding resource conflicts in intelligent agents. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)*, pages 18–22. IOS Press, 2002.

[van de Poel, 2013] Ibo van de Poel. Translating values into design requirements. In Diane P. Michelfelder, Natasha McCarthy, and David E. Goldberg, editors, *Philosophy and Engineering: Reflections on Practice, Principles and Process*, pages 253–266. Springer Netherlands, Dordrecht, 2013. doi:10.1007/978-94-007-7762-0_20.

[van den Hoven, 2007] Jeroen van den Hoven. ICT and value sensitive design. In Philippe Goujon, Sylvian Lavelle, Penny Duquenoy, Kai Kimppa, and Véronique Laurent, editors, *The Information Society: Innovation, Legitimacy, Ethics and Democracy", University of Namur, Belgium 22–23 May 2006*, pages 67–72. Springer US, Boston, MA, 2007. doi:10.1007/978-0-387-72381-5_8.

[Visser *et al.*, 2016] Simeon Visser, John Thangarajah, James Harland, and Frank Dignum. Preference-based reasoning in BDI agent systems. *Autonomous Agents and Multi-Agent Systems*, 30(2):291–330, 2016. doi:10.1007/s10458-015-9288-2.