Artificial Intelligence 175 (2011) 1180–1193

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Relational preference rules for control ☆

Ronen I. Brafman

Department of Computer Science, Ben-Gurion University, PO Box 653, Beer-Sheva 84105, Israel

ARTICLE INFO

Article history:
Received 28 February 2009
Received in revised form 4 July 2010
Accepted 4 July 2010
Available online 1 December 2010

Keywords:
Preference models
Preference rules
Relational models
Rule-based systems
Command and control automation

ABSTRACT

Value functions are defined over a fixed set of outcomes. In work on preference handling in AI, these outcomes are usually a set of assignments over a fixed set of state variables. If the set of variables changes, a new value function must be elicited. Given that in most applications the state variables are properties (attributes) of objects in the world, this implies that the introduction of new objects requires re-elicitation of preferences. However, often, the user has in mind preferential information that is much more generic, and which is relevant to a given type of domain regardless of the precise number of objects of each kind and their properties. Such information requires the introduction of relational models. Following in the footsteps of work on probabilistic relational models (PRMs), we suggest in this work a rule-based, relational language of preferences. This language extends regular rule-based languages and leads to a much more flexible approach for specifying control rules for autonomous systems. It also extends standard generalized-additive value functions to handle a dynamic universe of objects. Given any specific set of objects this specification induces a generalized-additive value function over assignments to the controllable attributes associated with these objects. We then describe a prototype of a decision support system for command and control centers we developed to illustrate and study the use of these rules.

1. Introduction

Much of the work in AI on preference handling has focused on tools for modeling preferences of lay users, often in applications related to electronic commerce, such as support for online selection of goods [29,12,10,4], tools for preference elicitation in combinatorial auctions [33], recommender systems [12], etc. Some work also targets the more classical decision-analysis setting which is usually mediated by an expert decision analyst, supporting the elicitation process of the detailed classical structures used there, namely utility functions (e.g., [11,17]). However, much less work considers the use of preferences as a key tool in the design of complex systems.

The idea of using preferences to design autonomous systems is quite intuitive. Autonomous systems make many decisions during their run-time, and ideally, their choices should be the ones maximally preferred among available choices at the current context. A preference-based design explicitly models the designer's preferences for different choices in different contexts, and uses a generic mechanism for selecting a preferred feasible choice at run-time.

A preference-based design can provide a uniform declarative and modular approach for the design and specification of certain autonomous systems. It is naturally amenable to customization, both before and during deployment, either by providing additional information about the context, or allowing for additional user-specific preferences. Compared with electronic commerce-based applications which deal with users who usually spend little time with the system and require

☆ A preliminary version of this paper appears in the Proceedings of the International Conference on Knowledge Representation and Reasoning, 2008 [9].
E-mail address: brafman@cs.bgu.ac.il.

0004-3702/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2010.11.010

an interface that is immediately intuitive, the design context allows for more sophisticated and rich methods. A system designer is likely to be willing to spend more than a few minutes on her system, and she can be expected to spend time learning how to effectively specify her preference. On the other hand, the designer's willingness to adopt a new tool is likely to depend greatly on the convenience and intuitive appeal of this tool, and on the amount of learning required to use it effectively without expert assistance. This puts the system design context somewhere between the end-user context and the decision analysis context, and motivates the need for formalisms that address this setting. These formalisms must provide sufficient expressiveness while remaining intuitive.

Decision-theoretic agent design is not a new paradigm. It pervades the classic text of [32], and recent work in robotics shows that it can be very successful [24]. A full-fledged decision-theoretic approach requires maintaining a probabilistic state estimate and a utility function. The main drawback of using classical, propositional probability distributions and utility functions is that they limit the agent's knowledge to a fixed set of propositions or objects. Thus, more generic knowledge about certain classes of objects, or relationships, cannot be captured. This limits the applicability of these systems to a single fixed, static domain. On the probabilistic side, relational and object-oriented probabilistic models provide tools that allow designers to describe generic probabilistic models that can then be used in diverse contexts in which the number and properties of concrete object instances may be quite different. On the preference side, we are not there yet, although the need to model preferences may be more pressing, as they are harder to learn from data because of their subjective nature.

This need for preference representation tools that support the system design and control context and provide the ability to express relational preferences in an intuitive manner that system designers can easily grasp, motivates this paper. Its main contribution is the introduction of a simple relational preference formalism whose semantics generalizes that of generalized additive value functions. In addition, it explains how optimal choices can be computed given such a specification using standard techniques, such as variable elimination, but also, how this problem can be reduced to the problem of computing the most probable explanation given a probabilistic relational model (PRM), leveraging existing algorithms for these models. Finally, we describe a concrete application domain, which is of independent interest, which serves to motivate this type of formalism and illustrate its possible application.

Relational Preference Rules (RPRs) specify preferences for systems that act in dynamic environments where both the set of objects and their state change constantly. They combine ideas from rule-based systems and earlier preference formalisms leading to a simple rule-based syntax with weights attached to different choices. The basic idea is very simple: for every value of a controllable attribute, specify what conditions affect its desirability, and how happy we would be to see this value in this context. The syntax is simple:

$$\bigwedge(\text{Condition on attributes other than } v) \rightarrow v : \langle \text{list of (weight,value of } v\text{) pairs} \rangle$$

Readers familiar with formalisms such CP-nets [6] and especially UCP-nets [5] will see the clear resemblance to the conditional preference/utility tables used there, with one main difference: the conditions expressed in those formalisms are propositional, whereas here we have conditions over relations. Indeed, given any concrete set of objects, these rules induce a concrete value function, which is induced by all possible groundings of these rules. Like rule-based systems, preference rules-based systems can be used in process and decision control applications in which rule-based systems are currently used. They retain the natural form of rule-based systems, but are much more flexible because their conclusions are not based on rigid deduction, but rather on optimization.

Preference rules bare certain resemblance to soft constraint logic programs (SCLP) [2], though their semantics is different. They are also closely related to generalized-additive value functions [16,1], as each ground rule is a factor in the induced value function. They can also be viewed as specifying a linear value function where the basis functions correspond to the rules. Similar rules have been suggested as a formalism to specify the behavior of a multi-agent system. And of course, these rules were motivated by PRMs, of which Markov Logic is the most similar, semantically [31]. We will discuss these relations in more depth in Section 4.

The rest of this paper is structured as follows: In Section 2 we describe and discuss the syntax and semantics of preference rules. Section 3 discusses the complexity of inference and how these rules can be transformed into Markov Logic theories. In Section 4 we discuss related work. In Section 5 we describe a system prototype we built using the methodology described in this paper. This system shows how preference rules can be used to select which information to display to decision makers in a real-time command and control center. We conclude with a discussion of future challenges in Section 6.

2. Preference rules

We introduce the syntax and semantics of preference rules, and follow up with a discussion of some of our choices.

2.1. The language

We adopt an object-oriented world model. Objects are instances of certain object classes. A set of attributes is associated with every instance of every class. The value of these attributes may be a simple type, such as integers, reals, strings, or an object class. Object-valued attributes capture binary relations between objects, and, in principle, any n-ary relation can be expressed using multiple binary relations. In addition, we may allow for class attributes. Attributes are separated into two classes: *controllable* and *uncontrollable*.

Example 1. Throughout the paper, we refer to the *fire-fighters* domain, which is an instance of the type of command and control application that motivated this work. In this domain, we are interested in supporting decision making in a command and control center for the emergency services of a large city. Our objects will correspond to objects of interest in this setting. Uncontrollable variables correspond to state variables (location of personal, fires, etc.), while controllable variables usually correspond to information sources we can turn on and off.

Some classes could be: *fireman, fire-engine, fire*, etc. *Fireman* might have attributes such as *location, rank*, and *role*, and a class attributes *base-station*. Imagine that in addition, firemen are equipped with sensors, such as a *camera*, *CO_2-level*, and *temperature*. *Fire* can have attributes *location* and *intensity*. *Fire-engine* might have attributes such as *location, driver, ladder*. The firemen's sensor attributes, as well as the *driver* and *ladder* attributes are themselves objects. The camera object has two attributes: *on*, and *display* which determine whether it is on, and whether the video stream is being displayed in the command center. Both of these attributes are examples of controllable attributes. We can imagine an application where the fire-engine's *driver* attribute is controllable, as well.

Our goal is to define a generic value function, one that can be used across different instances of the same domain type. Technically, we specify a function that, given a set of objects returns a value function over the set of possible assignments of the relevant attributes of these objects. This is done by using *preference rules* that take the following form:

$$\text{rule-body} \;\rightarrow\; \text{rule-head}: \big\langle (v_1, w_1), \ldots, (v_k, w_k) \big\rangle$$

Intuitively, the rule-head specifies a controllable attribute, and this rule specifies the desirability (the w_i's – weights) of different assignments of values (the v_i's) in the context specified by the rule-body. Thus, the specification process requires the designer to contemplate about what different contexts influence the desirability of different assignments to the possible values of a controllable variable, and how.

More precisely, *rule-body* has the following form:

$$\text{class}_1(x_1) \wedge \cdots \wedge \text{class}_k(x_k) \wedge \alpha_1 \wedge \cdots \wedge \alpha_m$$

and α_i has the form: x_i*.path REL value* or x_i*.path*$_i$ *REL* x_j*.path*$_j$. Each x_i must appear earlier within a $\text{class}_j(x_i)$ element. By *path* we mean a possibly empty attribute chain such as `x.mother.profession` and *REL* denotes a relational operator such as $=, \neq, >, <$, etc. The *rule-head* has the form x_j*.path* where x_j*.path* denotes a controllable attribute. $\langle (v_1, w_1), \ldots, (v_k, w_k) \rangle$ is a list of pairs, the first of which denotes a possible value of the attribute in *rule-head*, and the second of which is a real-valued weight. Given a rule r, we use $w(r, v)$ to denote the weight associated with assigning value v to the head of rule r. Thus, what we get is something akin to a weighted Horn rule, though we do not require that literals in the body be positive. Finally, we do not allow multiple controllable attributes within one attribute chain. For example, `x.girl-friend.salary` would not be allowed if one can both choose one's girl-friend, and the girl-friend's salary.

Example 2. The following rule expresses the fact that viewing the stream generated by a fireman in a location of a fire has value 4:

$$(1)\ \text{fireman}(x) \wedge \text{fire}(y) \wedge x.\text{location} = y.\text{location} \rightarrow x.\text{camera.display}: \big\langle (\text{“on”}, 4), (\text{“off”}, 0) \big\rangle$$

Note that this will have the same effect as

$$\big(1'\big)\ \text{fireman}(x) \wedge \text{fire}(y) \wedge x.\text{location} = y.\text{location} \rightarrow x.\text{camera.display}: \big\langle (\text{“on”}, 4) \big\rangle$$

Here is another rule that expresses a preference that the rank of a fireman whose camera is on will be high (e.g., so that we know what commanders on-site are seeing):

$$\text{fireman}(x) \wedge x.\text{camera.display} = \text{“on”} \rightarrow x.\text{rank}: \big\langle (\text{“high”}, 4) \big\rangle$$

However, this rule is not allowed, as the variable at the head — the rank of the fireman, is not directly controllable.

Here is another rule that would increase the value of observing the oxygen level of firemen in areas with a high level of CO_2:

$$(2)\ \text{fireman}(x) \wedge x.\text{CO}_2\text{-level} = \text{high} \rightarrow x.\text{oxygen-level-display}: \big\langle (\text{“on”}, 10), (\text{“off”}, 0) \big\rangle$$

Here is a another rule that includes a controllable attribute in the body:

$$(3)\ \text{fireman}(x) \wedge \text{fire-engine}(y) \wedge x.\text{location} = y.\text{location} \wedge y.\text{camera.display} = \text{on}$$
$$\rightarrow x.\text{camera.display}: \big\langle (\text{“off”}, 8), (\text{“on”}, 0) \big\rangle$$

It expresses the fact that we usually don't need another camera on when the fire-engine's camera is on in the same location.

Our need for a relational approach to preference specification stems from the fact that the set of objects is not known at design time. In fact, this set can change throughout the life-time of an application, and we need a fixed preference specification formalism that can work in diverse settings. For instance, new fires may occur, while others may be extinguished, and new equipment or fireman may be added. Moreover, we would not want to have to rewrite the logic behind the system each time we deploy it in a new location.

A note on notation. Although we use the term *relational rules*, we chose to use an object-oriented notation, rather than a pure relational one. In the application we worked on when designing these rules, described in Section 5, we found this notation to be more intuitive. This, of course, is a matter of taste, and one can switch between different notations.

Example 3. Here are rules 1 and 2, described using a relational syntax.

$$(1'')\ \text{fireman}(x) \wedge \text{fire}(y) \wedge \text{co-located}(x, y) \rightarrow \text{camera-display-on}(x) : \big\langle (\textit{true}, 4) \big\rangle$$

$$(2'')\ \text{fireman}(x) \wedge \text{high-CO}_2\text{-level}(x) \rightarrow \text{oxygen-level-display-on}(x) : \big\langle (\textit{true}, 10) \big\rangle$$

Early work in PRMs made a similar choice [25], but later work has adapted the relational notation. Indeed, the object-based notation is inconvenient when we wish to specify relations involving more than 3 objects. Binary relations, such as X is the commander of Y, can be captured in relational syntax using *commander*(X, Y) or using object-oriented notation as $X.\textit{commander} = Y$. However, a ternary relation such as *parents*(X, Y, Z), capturing the fact that X and Y are the parents of Z and X and Y are married, is hard to represent directly using an object-oriented notation. Of course, any k-ary relation can be represented using k binary notation, but this new representation is less intuitive and requires the introduction of additional “relation” objects. On the other hand, in decision-theoretic applications, it is natural to make the distinction between controllable and uncontrollable attributes. One could make a similar distinction between controllable and uncontrollable relations, but that is much less intuitive. For example, it is more natural to say that we can control the location of a fireman, rather than the relation *co-located*.

2.2. The semantics

Preference rules have the flexibility to model diverse settings because they are basically schemas of ground rules, and the concrete set of ground rules generated depends on the nature of the actual objects. A set of objects induces a set of ground rules. A ground rule instance is obtained by assigning a concrete object from the appropriate class to the rule variables. Thus, given a rule

$$\text{class}_1(x_1) \wedge \cdots \wedge \text{class}_k(x_k) \wedge \alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow \alpha$$

and an assignment of objects $o_1, \ldots, o_k$ to variables $x_1, \ldots, x_k$ from appropriate classes (i.e., o_i belongs to class$_i$), we obtain a ground rule instance which has the form:

$$\alpha'_1 \wedge \cdots \wedge \alpha'_m \rightarrow \alpha',$$

where α'_i is obtained from α_i by replacing each x_j by the corresponding o_j, and similarly for α'.

Example 4. Suppose that we have a single fireman, Alice, and a single fire-engine, Fred. Because there are no *fire* objects, there are no ground instances of Rule 1. However, Rules 2 and 3 have a single ground instance, each:

$$\text{Alice.CO}_2\text{-level} = \text{high} \rightarrow \text{Alice.oxygen-level-display} : \big\langle (\text{“on”}, 10), (\text{“off”}, -10) \big\rangle$$

$$\text{Alice.location} = \text{Fred.location} \wedge \text{Fred.camera.display} = \text{on} \rightarrow \text{Alice.camera.display} : \big\langle (\text{“on”}, 0), (\text{“off”}, 8) \big\rangle$$

Now, suppose that we add a *fire* object: Fire1. Rule (1) would have a single ground instance:

$$\text{Alice.location} = \text{Fire1.location} \rightarrow \text{Alice.camera.display} : \big\langle (\text{“on”}, 4), (\text{“off”}, 0) \big\rangle$$

If we add another fireman, Bob, then Rules (1)–(3) have another ground instance, as above, but with Alice replaced by Bob.

A set $\mathcal{O} = o_1, \ldots, o_n$ of objects also induces a set $\mathcal{A}_\mathcal{O} = \{A_1, \ldots, A_m\}$ of attribute instances with their respective domains. These are precisely the attributes associated with the objects in $\mathcal{O}$. Naturally, if objects are added or removed, then $\mathcal{A}_\mathcal{O}$ changes. We use $\bar{a}$ to denote a particular assignment to these attribute instances. Here we introduce an important *closed-world assumption*, which basically states that $\mathcal{O}$, the set of objects, is the entire set of objects. Thus, object-valued attributes (which can act like functions, or binary relations) must have a value in $\mathcal{O}$. This bounds the size and number of possible interpretations.

A preference rule base $\mathcal{R} = \{r_1, \ldots, r_k\}$, where each r_i is a preference rule, and a set of objects $\mathcal{O}$, induces a value function $v_\mathcal{O}$ over the possible assignments to $\mathcal{A}_\mathcal{O}$. The basic idea is straightforward: the set of objects induces a set of ground rules, and these induce the value function as follows: for each assignment, we sum the values associated with the ground rules it satisfies.

More formally, given an assignment to all attributes of all objects, we can evaluate all conditions of the form:

$$\alpha_1 \wedge \cdots \wedge \alpha_m \rightarrow x.path : \big\langle (v_1, w_1), \ldots, (v_k, w_k) \big\rangle$$

Let $\bar{a}$ be a possible assignment to the current objects' attributes. We say that r is *satisfied* by attribute assignment $\bar{a}$ if $\alpha_1, \ldots, \alpha_m$ are satisfied by $\bar{a}$. Recall that α_k has the form $x_i.path$ REL *value* or $x_i.path$ REL $x_j.path$, where x_i, x_j are object instances, so the definition of its satisfaction given a complete assignment to all attributes is straightforward.

Given a ground rule r and an attribute assignment $\bar{a}$, the rule's *value* given $\bar{a}$, denoted $w(r, \bar{a})$ is defined as follows:

- w_i if r is satisfied and $\bar{a}$ assigns v_i to $x.path$
- 0 if r is not satisfied or the value $\bar{a}$ assigns to $x.path$ is not in $\{v_1 \ldots v_k\}$

The value function $v_{\mathcal{R},\mathcal{O}}$ induced by a rule-base $\mathcal{R}$ on a set of objects $\mathcal{O}$ is simply the function that assigns to every possible attribute assignment $\bar{a}$ the sum of values of all its ground instances given $\bar{a}$.

$$v_{\mathcal{R},\mathcal{O}}(\bar{a}) = \sum_{\text{ground instances } r' \text{ of } r \in \mathcal{R}} w(r, \bar{a})$$

When $\mathcal{R}$ and/or $\mathcal{O}$ are fixed by the context, we shall omit these subscripts and simply write $v(\bar{a})$.

Example 5. Consider the rule-base $\mathcal{R}$ containing Rule (1) and a set of objects $\mathcal{O}$ containing fireman Alice and Bob, and a fire Fire1. The rules have two ground instances:

$$\text{(1a) Alice.location} = \text{Fire1.location} \rightarrow \text{Alice.camera.display} : \big\langle (\text{“on”}, 4), (\text{“off”}, 0) \big\rangle$$

$$\text{(1b) Bob.location} = \text{Fire1.location} \rightarrow \text{Bob.camera.display} : \big\langle (\text{“on”}, 4), (\text{“off”}, 0) \big\rangle$$

Given these ground rules, the value of an assignment to the attributes of *Alice, Bob, Fire1* depends only on their locations and the value of their *camera.display* attribute. If we have one fireman in the location of the fire and we display his camera, the value is 4. If we have two firemen in the location of the fire then the value is 4 if we display a single camera, and 8 if we display both. Under all other assignments, the value is 0.

2.3. Discussion

We now consider a number of issues that our formalism gives rise to. The first issue has to do with multiple rules with the same head. Imagine that we add the rule:

$$\begin{aligned}\text{(4) fireman}(x) \wedge \text{fireman}(y) \wedge \text{fire}(z) \wedge x \neq y \wedge x.\text{location} = y.\text{location} \wedge x.\text{location} = z.\text{location} \wedge \\ x.\text{camera.display} = \text{“on”} \rightarrow y.\text{camera.display} : \big\langle (\text{“on”}, -4) \big\rangle\end{aligned}$$

Intuitively, this rule says that there is a negative value to more than one camera at a location. We now have two rules, (1) and (4), that have the same head and can be applied in the same situation (i.e., there are object and attribute value choices in which the bodies of both rules are satisfied). Notice that with the introduction of this rule, it is less preferred to have two cameras on in the location of the same fire. For each camera, we get +4 from rule (1), but we also have two possible, symmetric instantiations of rule (4) that contribute −8. Altogether the value of turning two cameras on is 0, whereas each of the two assignments in which we turn one camera on and the other off, has a value of 4.

Should we allow multiple rules with the same head attribute? Our current formalism remains well defined in this case — it implies that we will add the contribution of the groundings of both rules. This implies that the "value" of a certain attribute value can be spread out among multiple rules, as opposed to a single place. A similar choice arises in PRMs. Bayesian Logic [21], for instance, allows multiple rules with the same head, while Relational Bayesian Networks [19] require unique rule-heads. In both cases, combination rules (whether within rules or external to them) are used to determine the final value. A combination rule tells us how to combine multiple influences. That is, how to infer $P(a|b,c)$ from $P(a|b)$ and $P(a|c)$. For example, consider the rule:

$$P\big(worried(x) = true | call(y, x) = true\big) = 0.9$$

This rule states that the probability that x will be worried given that x received a call from y is 0.9. Thus, $P(worried(a) = true|call(b, a)) = 0.9$, and $P(worried(a) = true|call(c, a)) = 0.9$. But what about $P(worried(a) = true|call(b, a), call(c, a))$? We don't really have information how to compute this value. One needs to explicitly state how to combine the two influences, e.g., something like:

$$combining\text{-}rule(worried) = noisy\text{-}or$$

(where noisy-or is a well known technique for combining multiple influences).

Using this terminology, in RPR, we have opted for summation as a single, default combination rule. To see this, suppose we have:

$$\mathit{person}(x) \wedge \mathit{neighbor}(x) \wedge \mathit{call\text{-}me}(x) \rightarrow \mathit{worried}\big\langle(\mathit{true}, 4), (\mathit{false}, 0)\big\rangle$$

then, if we have a context with one person who called me, the value is 4, and if two people call me, the value is 8, and more generally, this rule will contribute $4 \cdot k$, to assignments where *worried* is *true*, where k is the number of neighbors who called me. We believe that summation, or additivity, makes sense in the case of value functions. It does, however, assume some form of independence, and so there may be cases where it is not as natural. (For example, I no longer make distinctions between 3 or more calls from neighbors.)

Note that using this property, we were able to capture the fact that two cameras' displays from the same location are not as good as a single camera's display. In that example, we achieved it by using negative weights. But is the use of negative weights necessary? Because the model is additive, one is tempted to think that we can maintain the same relative order among alternative assignments if we add a constant c to the weight associated with every possible value of the attribute at the head. And if c is sufficiently large, all weights would be non-negative. This is true if no controllable value appears in the body of a rule. In that case, the number of times that a particular rule is fired is fixed by the assignment to the uncontrollable variables. Thus, the value of all assignments to the controllable variables increases by a constant (the number of times the rule is fired *times* c). However, if a controllable attribute appears in the body of some rule, this rule may be fired a different number of times in different assignment to the controllable variables. Thus, the change in value across different assignments will not be constant, and the ordering may not be preserved.

Example 6. Consider the following two rules:

$$\mathrm{class}_1(o) \wedge o.a = \mathit{true} \rightarrow o.b\,\big\langle(\mathit{true}, 0), (\mathit{false}, 4)\big\rangle$$
$$\mathrm{class}_1(o) \rightarrow o.a\,\big\langle(\mathit{true}, 4), (\mathit{false}, 0)\big\rangle$$

Here is the value associated with each assignment, assuming a single object:

	$a = T$	$a = F$
$b = T$	4	0
$b = F$	8	0

Now, suppose that we add a constant of −10 to the first rule, obtaining:

$$\mathrm{class}_1(o) \wedge o.a = \mathit{True} \rightarrow o.b\,\big\langle(\mathit{True}, -6), (\mathit{false}, -10)\big\rangle$$

The new values are:

	$a = T$	$a = F$
$b = T$	−2	0
$b = F$	−6	0

which clearly order the valuations differently.

This sensitivity to the actual weight values is a weak point of our formalism. However, we note that this is also a property of Markov Logic [31], which appears to be the most popular formalism for specifying PRMs. Other PRMs do not have this property, and pay for that in conceptual complexity and in the need to define explicit combination rules. One possible way to address this problem is by using learning algorithms that adjust rule weights based on experience. This can be done using existing reinforcement learning algorithms, and some positive preliminary results appear in [35]. Such algorithms allow for reduced sensitivity to exact weight values, and make the elicitation process simpler.

3. Finding an optimal assignment

Each set of objects has a number of attributes that are controllable, while the rest are not. The uncontrollable attributes can be viewed as specifying the context in which we operate. Our main computational task given a rule-base $\mathcal{R}$ and a set of objects $\mathcal{O}$ is to find an assignment to the controllable attributes that is optimal. Sometimes, as in our application, our choice is constrained. More formally, let $\mathcal{U}$ and $\mathcal{C}$ denote the uncontrollable and the controllable attributes in the set of attributes $\mathcal{A}$ induced by $\mathcal{O}$ and $\mathcal{R}$, respectively. Let $C \subseteq \Pi_{X\in\mathcal{C}} Dom(X)$ denote the set of feasible, or allowed, assignment to $\mathcal{C}$.[1]

Let $\mathbf{u}, \mathbf{c}$ denote assignments to the set of attributes $\mathcal{U}, \mathcal{C}$, respectively. Our goal is to compute:

$$\max_{\mathbf{c}\in C} v_{\mathcal{R},\mathcal{O}}(\mathbf{u}, \mathbf{c})$$

where $\mathbf{u}$ denotes the current context of uncontrollable attributes. That is, we want to find an assignment to the controllable attributes that maximizes v subject to constraints C and assignment $\mathbf{u}$.

There are a number of standard methods for solving such a problem. These include various variants of stochastic local search and systematic, branch and bound, search. We will not discuss the specifics of such methods, as they are well known, and our paper does not attempt to empirically evaluate different alternatives. Instead, we will discuss two methods. The first is variable elimination, which is also a standard method, but which is helpful in establishing some complexity results. The second is a reduction of this problem to a related optimization problem in PRMs. The main advantage of this latter transformation is that for these models a number of lifted inference techniques exist (i.e., ones that do not require grounding the rules) and we can immediately benefit from any new techniques in this area.

We note that, in practice, and especially for the application described in Section 5, the more standard methods may be better, and, indeed, we implemented them in our system. In that application, this optimization problem needs to be resolved whenever the value of the uncontrollable variables (i.e., the assignment $\mathbf{u}$) changes and a solution that is as close to the previous solution is preferred. In the case of stochastic local search, by starting the search from the current solution, we can bias the search to its vicinity. In the case of branch and bound, the current solution may provide a good initial lower bound.

3.1. Grounding and filtering

To determine the preferred assignment in a given context, consisting of an assignment to all uncontrollable variables, most standard algorithms require that first we ground and filter the rules.[2] This step generates the set of applicable ground rule instances and filters them according to the values of the uncontrollable variables.

Let r be a rule of the form

$$\text{class}_1(x_1) \wedge \cdots \wedge \text{class}_k(x_k) \wedge \alpha_1 \wedge \cdots \wedge \alpha_m \to \alpha : \langle (v_1, w_1), \ldots, (v_k, w_k) \rangle$$

Let $\mathcal{O}$ be the current set of objects with their uncontrollable values fixed.

(1) The rule grounding and filtering process considers all possible assignments to $x_1, \ldots, x_k$ from classes $\text{class}_1, \ldots, \text{class}_k$, respectively.
(2) It replaces each occurrence of an attribute by its value given the current assignment.
(3) It evaluates any condition α_i that depends only on the value of uncontrollable attributes.
    (a) If a condition evaluates to *false*, this grounding is ruled out.
    (b) Conditions that evaluate to *true* are dropped.

The result of this filtering stage is a set of ground rules involving only controllable attributes of the form:

$$\psi_1 \wedge \cdots \wedge \psi_l \to \psi \langle (v_1, w_1), \ldots, (v_k, w_k) \rangle$$

where each ψ_i has the form $o.path = r$, or $o.path = o'.path'$ where $o, o' \in \mathcal{O}$ $o.path, o'.path'$ denote an attribute path starting at o and o', ending in some controllable attribute, and r is a value of some simple type. ψ has the form $o.path$, and w_i is the weight associated with value v_i, which is identical to the weight of v_i in the original rule. We have actually seen examples of this process, e.g., in Example 4.

3.2. Complexity of grounded optimization

Finding an optimal assignment given a set of ground preference rules and objects is NP-hard both for the unconstrained problem and, obviously, for the constrained problem.

Theorem 1. *Deciding whether an assignment to the controllable attributes is optimal given a set of grounded, filtered preference rules is NP-hard.*

Proof. This result follows immediately from classical results on the complexity of max-Horn-sat, i.e., the problem of deciding whether a truth assignment is maximal with respect to a set of weighted Horn clauses, which is NP-hard [20]. To see

[1] Here we are agnostic as to the language specifying the constraints, although this choice can affect the complexity of the constrained optimization task.
[2] The exception would be lifted inference algorithms which exist for PRMs.

this, note that every weighted Horn-rule of the form $\{\text{body} \to \text{head} = v : w\}$ is equivalent to a ground rule of the form: $\text{body} \to \text{head}\langle(v, w)\rangle$. Thus, our decision problem is at least as hard as max-Horn-sat. We note that given certain properties of the constraint graph induced by the rules, this problem can be solved in polynomial time, as we discuss below.

It is also instructive to see another proof by reduction from the problem of deciding whether an assignment is the most probable explanation (MPE) to a Bayesian network. This highlights the important relation between inference in our model and in probabilistic models. MPE is known to be NP-hard [36]. Given a Bayes-net over variables $X_1, \ldots, X_n$, define a single class *class* with attributes $X_1, \ldots, X_n$ and a rule for every entry in the CPT of each variable. (We ignore the object from now on, as there is a single one in each rule.) That is, for every variable X_i and for every possible assignment pa_i to its parents in the Bayes-net. We add the rule:

$$pa_i \to X_i\big\langle\big\{(x_i, \exp\big(Pr(x_i|pa_i)\big) \bigm| x_i \in \mathcal{D}(X_i)\big\}\big\rangle$$

The size of the set of rules is linear in the size of the Bayes-net. Recalling that the joint probability in a Bayes-net is the product of the local conditional probabilities, and that the value in our formalism is obtained by adding the weights over all applicable rules, and because

$$\arg\max \sum w_i = \arg\max \prod \exp(w_i)$$

we conclude that this is, indeed a reduction (and obviously, a polynomial one).

This transformation between computing MPE and finding the most preferred assignment to a generalized-additive value function is well known. Here, the multiplicative factoring of probabilistic models leads to an additive factoring of their logarithm, which can be viewed as a generalized additive value function. □

Of course, our original problem is given by a set of relational rules, not the grounded filtered rules. The grounded, filtered set of rules can be at most exponential in the maximal number of variables appearing in each rule, and linear in the number of rules.

3.3. Variable elimination

A set of ground rules defines a value function. However, this value function has special structure. It is a sum of sub-functions, each one corresponding to a single rule. Each such rule usually refers to a small number of attributes. This form of a value function $v(\bar{X}) = \sum v_j(\bar{Z}_j)$, where $Z_j \subseteq X$, is known as a generalized additive value function (or GAI for short).

For example, consider the following ground rule instance:

$$\text{Alice.location} = \text{Fire1.location} \to \text{Alice.camera.display} : \big\langle(\text{“on”}, 4), (\text{“off”}, 0)\big\rangle$$

This defines a factor over the variables *Alice.location*, *Fire1.location*, and *Alice.camera.display*. This factor assigns a value 4 to all cases in which Alice.location = Fire1.location and Alice.camera.display = “on”, and 0 otherwise. Variable elimination is a generic technique that can be used, among other things, to compute a maximizing assignment. The basic idea is to push the max operator forward, as much as possible based on the following equivalence:

$$\begin{aligned}&\max_{x_1}\ldots\max_{x_n}\big[f_1(\cdot) + \cdots + f_k(\cdots) + f_{k+1}(\cdots, x_n) + \cdots + f_j(\cdots, x_n)\big]\\ &\quad = \max_{x_1}\ldots\max_{x_{n-1}}\Big[f_1(\cdot) + \cdots + f_k(\cdots) + \max_{x_n}\big[f_{k+1}(\cdots, x_n) + \cdots + f_j(\cdots, x_n)\big]\Big]\end{aligned}$$

where $f_1, \ldots, f_k$ are functions that do not depend on x_n, whereas $f_{k+1}, \ldots, f_j$ are functions that do depend on x_n. Defining

$$g_n(x_1, \ldots, x_{n-1}) = \max_{x_n}\big[f_{k+1}(\cdots, x_n) + \cdots + f_j(\cdots, x_n)\big]$$

we now have a new maximization problem, where x_n was eliminated:

$$\max_{x_1}\ldots\max_{x_n}\big[f_1(\cdot) + \cdots + f_k(\cdots) + f_{k+1}(\cdots, x_n) + \cdots + f_j(\cdots, x_n)\big] = \max_{x_1}\ldots\max_{x_{n-1}}\big[f_1(\cdot) + \cdots + f_k(\cdots) + g_n(\cdots)\big]$$

For more information on this well know technique, see, e.g., [15].

In the worst case, the complexity of this technique is exponential in the tree-width of an appropriate graph. The graph in question, which is also known as a *cost network* [15], contains a node for each ground attribute, and an edge between any two ground attributes that appear in the same rule. Its tree-width is often much smaller than the number of nodes. For example, imagine that each relational rule contains reference to a single object instance only, i.e., it has a single variable, and different rules refer to different object classes. In that case, the tree-width of the graph is bounded by the maximal number of attributes per rule.[3]

[3] Of course, this is an obvious case in which the problem is much simpler. The point is that variable elimination can benefit from this and other types of loose coupling between variables, automatically.

3.4. Reduction to PRMs

Earlier, we noted some similarities between our model and PRMs. Now, we show how to reduce the problem of optimal action choice in using RPRs to the problem of computing the most probable explanation (MPE) for a Markov Logic Network [31], one of the most popular current PRMs.

A Markov Logic network consists of a set of weighted first-order formulas. Together with a set of constants, it defines a Markov network with one node per ground atom and one feature per ground formula. The weight associated with a feature (generally, a real-valued function of the value of the nodes on which it is defined, here, a Boolean function of the relevant atoms) is the weight of the first-order formula from which the ground formula originates. The probability distribution over possible worlds x specified by the MLN and constants is thus $P(x) = 1/Z \exp(\sum_i w_i n_i(x))$, where w_i is the weight of the ith formula and $n_i(x)$ is its number of true groundings in x, and Z is a normalizing constant. Thus, this is basically a method for assigning weights to first-order models whose domain is restricted to a given set of constants.

Suppose we are given a preference rule of the form

$$\text{body} \rightarrow \text{head}\big\langle(v_1, w_1), \ldots, (v_k, w_k)\big\rangle$$

We can transform it as follows into a set of weighted formulas, as used in MLNs: $\{\text{body} \wedge \text{head} = v_i : w_i\}$. For example:

$$\text{fireman}(x) \wedge \text{fire}(y) \wedge x.\text{location} = y.\text{location} \rightarrow x.\text{camera.display} : \big\langle(\text{“on”}, 4), (\text{“off”}, 0)\big\rangle$$

would be transformed into:

$$\text{fireman}(x) \wedge \text{fire}(y) \wedge x.\text{location} = y.\text{location} \wedge x.\text{camera.display} = \text{on} : 4$$

and

$$\text{fireman}(x) \wedge \text{fire}(y) \wedge x.\text{location} = y.\text{location} \wedge x.\text{camera.display} = \text{off} : 0$$

To find the optimal assignment, we instantiate all variables denoting uncontrollable attributes, and are left with controllable variables only. Now, we compute the most-probable explanation, i.e., the assignment with highest probability:

$$\arg\max_{x \in X} P(x) = \arg\max_{x \in X} 1/Z \exp\left(\sum_i w_i n_i(x)\right) = \arg\max \sum_i w_i n_i(x)$$

The latter is, essentially, the value function associated with the original preference rules.

Using this reduction, we can benefit from any improvement in algorithms for solving MPE for MLNs. Until quite recently, inference in probabilistic models was done by first grounding the model and then applying propositional reasoning techniques. Recent work has come up with lifted algorithms for probabilistic reasoning. For MPE, we can use first-order variable elimination [28,14]. The key idea in this method is to do as much as the computation without grounding (hence “lifted”) the rules or formulas. In first-order variable elimination, instead of eliminating one variable at a time, as in regular variable elimination, one eliminates a collection of related variables all at once. Where variables are related if they are different instantiations of the same predicate (as in *tall(alice)*, *tall(bob)*, ...). This method is elegant, yet complex conceptually and computationally. And it is not yet clear that it is more efficient in practice then grounded methods. Still, theoretically, it can scale up much better with the number of objects.

4. Related work

There are many diverse formalisms for specifying preferences and real-valued functions over structured discrete domains (i.e., domains described using multiple discrete variable), many of which have relational versions. Here, we discuss those we find most relevant to RPRs.

Mathematically, a probability distribution is a value function with special properties. Hence, it is not surprising that the preference rules formalism was inspired by PRMs. Indeed, there is much similarity between utility functions, which capture preferences, and probability functions. Probability distributions have a natural multiplicative decomposition (via the chain rule), while value and utility functions often have an additive decomposition, very much like log probabilities. Because log and exp are monotonic functions, the same values that maximize with respect to p, maximize w.r.t. $\log(p)$. Of course, there are also important differences: the notion of a conditional distribution plays an important part in multiplicative decomposition via the chain rule and it has no apparent semantics in the case of value function. Still, some of the computational questions in preference rules and PRMs are similar.

One of the earliest attempts to combine first-order logic with probabilities and utilities is Poole's independent choice logic (ICL) [27]. This is a very general formalism that can model games, where different agents have different valuations to each state. Our formalism is geared more towards systems with multiple controllable variables, each of which can be viewed as an agent. But this system is cooperative, with a single value associated with every assignment. In ICL, the utility of a state is described via clauses of the form *utility*$(agent, value) : -$*condition*. Multiple clauses of this form for the same agent need to be consistent (i.e., same values under the same conditions) and complete (i.e., specify values for each condition). Preference rules, on the other hand, have an additive structure that applies to multiple groundings and to multiple rules with the same head. Instead, in ICL, the rules should partition the space of possible worlds/assignments and there is no built-in additive semantics. We will see that this is typically the difference between preference rules and similar formalisms, such as SCLP and Relational MDPs, described below.

RPRs also resemble soft constraint logic programs (SCLP) [2,3]. SCLPs are constraint logic programs in which standard constraints are replaced by soft-constraints. A program is a set of Horn clauses, where the head is an atom, and the body is a collection of atoms or a value from some set of values of some semi-ring. An interpretation maps each ground atom to a value, and it satisfies a clause if the value of the head is greater than or equal to the value of the body. It is a model of the program if it satisfies all of its clauses. As in logic programs, there is the notion of a minimal model associated with an SCLP, as well. A minimal model is obtained by intersecting all models of the program. Here, intersection implies taking the minimal value assigned to each atom, where minimality is defined w.r.t. the semi-ring used. Finally, a goal is simply a collection of atoms.

Intuitively, a minimal model should correspond to an optimizing value function. However, there are differences in the semantics that make it non-trivial to map between the two formalisms. The most important aspect is that of additivity. In preference rules we have two aspects of additivity. First, the value of a model is the sum of weights associated with all ground instances of the rules. In SCLP, the value of a non-ground formula is the supremum over the values associated with the ground instances. Similarly, when we deal with combinations, i.e., multiple rules with the same head, the semantics of SCLPs is again one of maximization rather than sum. Another difference is that values are not associated with every clause, but only with clauses without a body.

A popular model for sequential decisions is that of a Markov Decision Process (MDP) [30]. Relational variants of MDPs were introduced by [7]. A relational MDP is a more fundamental model than RPRs in the sense that it describes the underlying dynamics of the world. RPRs were devised, primarily, for contexts in which there is no need or no ability to specify an underlying world model. For example, in the application presented in this paper, it is hardly likely that we have the knowledge to specify a good world model describing the dynamics of rescue and emergency services. However, domain experts have good intuitions about what behavior is good and what behavior is bad, and we seek to capture this in a convenient manner.

In MDPs, the term *value* function is associated with a specific policy, and maps possible states to their value given this policy. An optimal value function maps each state to its value given an optimal policy. In [7,34], reward and value functions for relational MDPs are defined by a case statement. That is, by partitioning the set of possible states according to certain properties, and associating a value to each part. These properties are described via sentences in the situation calculus. This is a much simpler and more explicit notation than preference rules. The former describes an explicit partition of the state space, whereas the latter is implicit, contains additive structure, cyclical references, and require counting the number of ground instances of rules. This is to be expected because the value function of an MDP specifies the value of states according to the optimal policy. Preference rules, on the other hand, can be viewed as specifying the value of *all* possible policies, and hence contain explicit reference to the choice of action/controllable variable.

Recently, [39] introduced first-order decision diagrams (FODD) as a more compact tool for representing relational value functions. FODDs are very much like BDDs and ADDs. A FODD is a labeled rooted directed acyclic graph, where each non-leaf node has exactly two children. The outgoing edges are marked with values true and false. Each non-leaf node is labeled with an atom $P(t_1, \dots, t_n)$, or an equality $t_1 = t_2$, where each t_i is a variable or a constant. Leaves are labeled with numerical values. An FODD is, thus, also a specification of a value function over states which partitions the set of states, but in a more compact manner than a case statement. A FODD could be used to specify a value function over the set of possible states and policies, by adding conditions that pertain to action values. However, its utility as a tool for specifying value functions is not clear because it is not an intuitive format for users to specify directly, nor is it clear that it gives computational advantages for optimization.

One popular approach for obtaining approximately optimal policies in large MDPs is linear value-function approximation [38,8]. Given a set of basis functions φ_i (defined on states), a vector of weights is sought such that $\sum_i w_i \cdot \varphi_i(\cdot)$ is a good approximation of the true value function. Two key issues are computing (or learning) good weight vectors and generating good basis functions. Our formalism can be viewed as defining a linear-value function over possible assignments where the basis functions, or features, correspond to rules:

$$\sum_i \sum_j c_{i,j} \phi_{i,j}(\bar{a})$$

Here $\phi_{i,j}(\bar{a})$ is the number of times that the ith rule is fired when the head of this rule is assigned its jth value in $\bar{a}$, and 0 otherwise; and $c_{i,j}$ is the weight associated with the jth value in this rule. Although we do not assume that our underlying model is an MDP, we believe that we can use reinforcement learning algorithms to improve rule weights and rule bodies. Successful application of these ideas will imply that designers need only specify reasonable rules with reasonable weights, and that learning algorithms can automatically improve them online. Encouraging initial results appear in [35].

We explained early on that we view preference rules as a good candidate for prescribing system control. For such applications, it is really only interesting if we have multiple, inter-dependent controllable variables. This is the case when we control a multi-agent system. Each variable could be viewed as describing the choice of action of an agent, or the role the agent takes on in a particular context. The value of a joint-strategy can be described using a set of rules. This idea was explored by [18]. Rules of the form $\langle \rho, c : v \rangle$, where c is some condition on the set of variables (the context and agent variables), v is a real value, and ρ is the function which assigns assignments in which c holds a value of v. The semantics is additive, and the value function is defined as the sum of contributions of all rules. This framework is defined abstractly, and the conditions can be formulated in propositional logic, or predicate calculus. Indeed, [22] applies these ideas to the robot soccer domain, where the rules are conjunctions of conditions expressed using predicates with free variables. Thus, like our relational rules, these are schemas that can be applied in contexts with different numbers of soccer playing agents. The optimal assignment is computed using variable elimination, and is used to control the system.

5. Preference rules for command and control display

Our work on preference rules arose out of our desire to solve a concrete problem for which we found current techniques inadequate. We explain this problem domain and describe a system prototype we built to test these ideas.

5.1. Problem domain

Consider a command and control room for real-time operations. This can be a control room for the fire-department of a large city; a control room for handling large-scale emergencies, such as earthquakes, flooding, or terrorist attacks; a control center for complex missions, such as NASA shuttle missions; an army command center; or a monitoring center for some firm that receives much real-time data that requires the attention of a decision maker. To be concrete, let us focus on the first setting, that of a fire department that needs to deal with fire and rescue operations, and disasters. Imagine, a not very futuristic scenario in which each fireman has a camera on his helmet, cameras are located in various places across the city (as is the case today in London and many other cities), and heat, smoke, and other hazardous material sensors are located on some firemen and in various places in buildings.

In a realistic scenario, we are talking about thousands of possible information streams that the decision maker in the control room might have access to. Moreover, much more information is accessible by querying databases: from street maps to building plans and building material specifications, to resource status such as firemen, fire trucks, special equipment, etc. Analysis tools that analyze the various aspects of the current situation, statistical tests, risk analysis, and simulations may also be available.

Clearly, a given decision maker cannot handle this volume of information. An ideal intelligent application will be able to fuse all the information together, but this is unlikely to happen in the foreseeable future. While much work is carried out on sensor fusion [23], current technology is very limited in the type and scope of sensory data that can be fused together. We seek a more modest solution, that although challenging, appears to be feasible, and can have enormous impact on the quality of decisions made. We seek a tool that will automatically control the information sources displayed to a decision maker based on the current state of the world.[4]

Consider the characteristics of such a system. The set of object types, i.e., classes, is known ahead of time. Some of the instances may also be known (e.g., the personnel in the fire department) but some change dynamically (e.g., a new fire is modeled as a new object of the *fire* class). The attribute values of concrete object instances change throughout the situation. As the state of the system changes (i.e., attribute value change or instances are added or removed), display choices must be made, and they must be made quickly (i.e., in a matter of seconds). Finally, information can be displayed in various modes, for instance, a video can be shown in different sizes and in different places on the user screen. Analysis results or interesting items of information can be shown directly, or using icons that require a click.

Attributes that represent display choices, such as whether a video stream is displayed and how, correspond to controllable attributes. All other attributes are uncontrollable. We seek a specification that describes the preferred values of controllable variables as a function of the values of uncontrollable variables and other controllable variables. For example, whether I want to show a video stream from the camera of a fireman in some fire scene depends on the fireman's status and rank, and on which other cameras will be shown. In some situations, one camera from a scene is enough, and in others more cameras are preferred.

Abstracting away a bit, we see an application where we must assign value to a number of controllable variables. The desirability of an assignment to a controllable variable depends on the state of the uncontrollable variables and the value assigned to other controllable variables. This set of variables changes dynamically as the set of objects changes. In addition, value choices are constrained by resource limitations (e.g., screen size, user attention). Using a simple rule-based system to determine the value of controllable variables is not natural and unlikely to be feasible as the resource constraints introduce strong inter-dependence into the system. A value function, on the other hand, provides context dependence in two ways. First, explicitly by allowing us to condition value on various choices and external parameters. Second, implicitly, via the ideas of constrained optimization, where we seek an optimal feasible solution.

[4] We do not address the difficult practical question of how the state of the world is maintained. In some situations, we can assume human operators update the database, such as 911 operators. In others, the state is automatically updated, as in corporate settings where online transactions contain all the relevant information.

5.2. System prototype

We built a system that is able to select among incoming video feeds and present the selected images on screen. Our system was configured to show three video files concurrently (although any other number can be selected), depending on the current state. The system selects both which video streams to show on the user's screen, and where. In addition, the system can run queries in the background and presents icons on the user's screen that provide quick access to the query result. The choice of which query to run is also context-dependent.

The system can function in live or simulated mode. All it needs is a list of available video streams, some event update feed which updates the state of the monitored environment, and the preference rules base. Simulation mode works in much the same way, but with an event file which stores events and their time of occurrence, and files storing video streams which are used in place of the live feeds. As far as the system is concerned there is no difference, as it expects URLs for the information sources as well as a feed of state updates.

As we do not have access to a real command and control center, we emulated video feeds by videos captured from the game *unreal tournament*.[5] In one of the modes of this game, two teams play a game of capture the flag. Each team must protect its own flag while attempting to capture the flag of its opponents. The system developers played the game and recorded the environment throughout the game. The recorded video streams correspond to each of the players (i.e., they constantly monitor the actions of this player) as well as each of the flags. These videos were stored in a database and were used to test the system in simulation mode. One such run of the system can be viewed at http://www.youtube.com/watch?v=UP37mlxWToA. This narrated demo is self explanatory. It shows how the display choices change as events occur, and explains the rationale behind these.

The system itself is quite flexible, although we tested it only on the capture-the-flag domain. It is implemented in JAVA. Documentation is available at http://www.cs.bgu.ac.il/~giorasch/documents.htm and the sources from the author. The system supports both stochastic local search and branch and bound algorithms. The demo referred to above uses stochastic local search and a system of objects containing 25 sensors that act as uncontrollable variables as well as 6 controllable variables (corresponding to cameras attached to the agents and the flags). The number of free variables in rules is at most 3, and branch and bound had no problem with this small domain.

Our system is only an initial prototype and many issues need to be addressed to make it more realistic. Our main goal was to demonstrate the process of selecting the appropriate cameras as a function of the current context. Much more attention to user interface issues is required. On the one hand, one would like to make transitions in display smooth and minimal. On the other hand, changes in display grab the user's attention, and sometimes this is desirable. The principled way of addressing this would be to change the objective function to reflect the (un)desirability of particular changes between consecutive configurations. In certain cases, this can be modeled using additional preference rules, for example:

$$\text{fireman}(x) \wedge x.\text{camera.display}' = \text{on} \rightarrow x.\text{camera.display}\big\langle(\text{on}, 0), (\text{off}, -1)\big\rangle$$

Here, x' refers to the previous value of attribute x, and we are basically penalizing a change in value. Of course, not all smoothness constraints can be naturally captured this way, but the general idea of amending the value function with smoothness preferences can be used to address diverse preferences, and existing optimization methods can be used to optimize the new function (although reduction to PRMs may not work now). Another important HCI issue is how to provide the user with the right balance of manual control and automated change in display.

A more fundamental technical issue is our reliance on a source that feeds us new events and reassigns all uncontrollable variables. In real systems this usually requires advanced image processing techniques. Moreover, we would expect that many important uncontrollable variables would not be perfectly observable, requiring us to deal with uncertainty about the context. Handling this requires a richer model, ideally, probabilistic, and the ability to compute and maximize the expected value of assignments to the controllable variables.

6. Summary and future work

We presented a new flexible approach to preference specification based on RPRs, which have a number of advantages. First, they are very intuitive. Second, they are much more flexible than regular rules — they are not rigid: their choices are sensitive to context and other choices made, and they can be augmented by external constraints while retaining their semantics. Preference rules can be specified for a generic system — i.e., given only knowledge of the set of classes, and for any set of objects they induce a generalized additive value function. Thus, they can be used to control dynamic systems in which both the state and the number of objects changes, as their specification does not require complete information about the system, only the classes of its basic components.

This work provides a rich set of questions for future work. The first question is the adequacy of the proposed model. In modeling the command and control setting we run into a number of issues that might suggest some revisions. Consider the example we used in which having one camera on in a certain arena was valuable, but any additional cameras were counter-productive. To capture this we gave a positive value to each camera that is on, and a negative value to each pair of

[5] http://www.unrealtournament3.com.

cameras that is on. Is there a more direct way of capturing this within a single rule, possibly by referring to the number of objects satisfying a rule? Similarly, in another domain we are currently working on, preference rules are used to describe the preferred behavior of agents within a computer game. Some of the state variables of these agents are integer-valued (e.g., their health level, their ammunition level). In this domain it seemed natural to make the weight associated with an assignment a function of some of these integer-valued variables, rather than a constant.

Returning to our current formalism, an important practical challenge is providing lifted inference. The number of ground rules quickly grows as the number of objects increases – exponentially in the number of free variables per rule. Lifted inference methods may be able to perform much of the computation off-line, or support methods that work with the original set of rules without generating ground rules. As we showed, we can map maximization over preference rules to a suitable maximization problem over PRMs, utilizing the results obtained in that area. This raises the issue of how to integrate relational preference and probability models, to provide a full decision-theoretic model. While the specification part is simple – we have relational probabilistic and utility models now – the main question is one of efficient inference. There are two different cases. The simpler case is one where preferences are defined on controllable variables, as in our current formalism, yet the value of the uncontrollable variables is unknown. In this case, assuming we have a probability distribution over possible values of the uncontrollables, we need to compute the expected value of assignments to the controllable variables. In that case, we can compute the probability of the uncontrollables in the body of each rule, scale the value accordingly, and perform optimization as usual. Note that with a suitable PRM, this can be done often without grounding the rules.

A more complex case is one where we have preferences over the uncontrollable variables, and their value is influenced stochastically by the assignment to the controllable variables. For example, in a soccer game we care about the final score, and our actions (such as kicking, or passing) have no inherent value. They do, however, influence the likelihood that we will win. In this case, the probabilities and values/utilities are more tightly coupled, and solving this maximization problem, especially without grounding all rules, is a real challenge.

An interesting theoretical question is the tradeoff between the succinctness of a formalism for behavioral specification (i.e., the size of the specification), the complexity of computing an optimal behavior, and the ease of elicitation. While the latter is hard to measure, except for via empirical evaluation, the relationship between the former two criteria is a staple of research in knowledge representation. The work that appears to be most relevant to this question is that of knowledge compilation. In particular, Darwiche and Marquis [13] consider the problem of compiling propositional weighted bases. Weighted bases are knowledge bases based on penalty logic [26]. They consist of pairs of the form $\langle \phi, w \rangle$, where ϕ is some propositional formula, and w is some integer-valued weight. The weight should be thought of as the cost of violating this formula. The value of a complete assignment (possible world) is the sum of weights associated with those formulas that are not satisfied. Hence, an optimal assignment is one that minimizes this value. The propositional analogue of our preference rules is equivalent to a weighted base of clauses. The key question in the theory of knowledge compilation is whether a knowledge base in one form can be compiled into a knowledge base in some other form off-line so that queries can be answered more efficiently with respect to the new knowledge base. Efficiency is defined with respect to the size of the original knowledge base, so that compiling to an exponentially larger knowledge base is not a useful approach. To define the problem more precisely, one needs to be explicit about the form of the knowledge-base (the static part) and the queries. The idea is that it could be worthwhile to spend much effort compiling the static part once, in order to answer many queries much more quickly, online.

Existing results on weighted knowledge bases cover the case of queries that either ask whether a particular possible world is minimal or whether a certain formula is implied by all minimal possible worlds. The knowledge-base is the static part and the possible assignment, or formula, are the dynamic part. Our problem is somewhat different. First, it is an optimization problem and not a decision problem: we seek an optimal assignment, as opposed to deciding whether a particular assignment is optimal. Second, the dynamic part in our case is the assignment to the uncontrollable variables. This means that the set of "active" clauses changes each time – thus, the methods described in [13] do not apply. In fact, it is not hard to build a set of rules that is polynomial in the number of uncontrollable variables, yet induces exponentially many different sets of active rules.[6]

Perhaps more relevant to this discussion is the work of [37]. That work can be viewed as exploring various variants of penalty logic that differ in the type of propositional formulas they allow. The authors present a large number of variants and compare their succinctness, expressivity, and complexity. It also considers the "right" complexity question from our perspective, i.e., that of finding an assignment of maximal utility. Their results provide an important piece in this puzzle.

However, ultimately, we would like to see a better understanding of this question in a wider scope. Formally, (focusing on Boolean variables and the propositional case) our specification language defines a mapping from $\{0,1\}^n$ to $\{0,1\}^m$, where n is the number of uncontrollable variables and m is the number of controllable variables. This function returns an optimal assignment to the controllables given a certain assignment to the uncontrollables. Thus, we must explore alternative ways of representing such functions, the difficulty of transforming one to the other, how large the output is with respect to the input, and how quickly can one compute this function given a certain representation. Note that, formally, such a function can be viewed as a set of m Boolean functions, where the latter is one of the most central structures in computer science.

[6] For every value of an uncontrollable variable, we will have a specific rule over controllable ones.

But while m Boolean functions may be a suitable target representation, it is quite different from our input representation which intentionally mixes the different controllable variables, unless each rule contains a single controllable variable – a degenerate and not very interesting case.

Acknowledgements

I am grateful to Manfred Jaeger for his help relating this work to PRMs, to Liron Himi, Ofer Schonberger, Yuval Shahar, and Giora Shcherbakov, for system development, to Yagil Engel for comments on earlier drafts. This work was supported in part by ISF grant 1101/07, the Paul Ivanier Center for Robotics Research and Production Management, the Lynn and William Frankel Center for Computer Science, and COST action IC0602.

References

[1] F. Bacchus, A. Grove, Graphical models for preference and utility, in: Proc. of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI'95), 1995, pp. 3–10.
[2] S. Bistarelli, Semirings for Soft Constraint Solving and Programming, Springer-Verlag, 2004, Chapter 6.
[3] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint logic programming, in: Proc. of the Fifteenth IJCAI (IJCAI'97), 1997, pp. 352–357.
[4] J. Blythe, Visual exploration and incremental utility elicitation, in: Proc. of the Eighteenth AAAI (AAAI'02), 2002, pp. 526–532.
[5] C. Boutilier, F. Bacchus, R.I. Brafman, UCP-networks: A directed graphical representation of conditional utilities, in: Proc. of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01), 2001, pp. 56–64.
[6] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, D. Poole, CP-nets: A tool for representing and reasoning about conditional *ceteris paribus* preference statements, Journal of Artificial Intelligence Research 21 (2004) 135–191.
[7] C. Boutilier, R. Reiter, B. Price, Symbolic dynamic programming for first-order MDPs, in: Proc. of the Seventeenth IJCAI (IJCAI'01), 2001.
[8] S.J. Bradtke, A.G. Barto, Linear least-squares algorithms for temporal difference learning, Machine Learning 22 (1–3) (1996) 33–57.
[9] R.I. Brafman, Relational preference rules for control, in: Proc. of the Eleventh International Conference on Knowledge Representation and Reasoning (KR'08), 2008, pp. 552–559.
[10] R.I. Brafman, C. Domshlak, T. Kogan, Compact value-function representations for qualitative preferences, in: Proc. of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04), 2004, pp. 51–58.
[11] D. Braziunas, C. Boutilier, Local utility elicitation in GAI models, in: Proc. of the Twenty First Conference on Uncertainty in Artificial Intelligence (UAI'05), 2005, pp. 42–49.
[12] L. Chen, P. Pu, Preference-based organization interface: Aiding user critiques in recommender systems, in: Proc. of the Eleventh International Conference on User Modeling (UM'07), 2007.
[13] A. Darwiche, P. Marquis, Compiling propositional weighted bases, Artificial Intelligence 157 (2004) 81–113.
[14] R. de Salvo Braz, E. Amir, D. Roth, Lifted first-order probabilistic inference, in: Proc. of the Nineteenth IJCAI (IJCAI'05), 2005, pp. 1319–1325.
[15] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[16] P.C. Fishburn, Utility Theory for Decision Making, John Wiley & Sons, 1970.
[17] C. Gonzales, P. Perny, GAI networks for utility elicitation, in: Proc. of the International Conference on Knowledge Representation and Reasoning (KR'04), 2004, pp. 224–234.
[18] C. Guestrin, S. Venkataraman, D. Koller, Context-specific multiagent coordination and planning with factored MDPs, in: Proc. of the Eighteenth AAAI (AAAI'02), 2002, pp. 253–259.
[19] M. Jaeger, Relational Bayesian networks, in: Proc. of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI'97), 1997, pp. 266–273.
[20] B. Jaumard, B. Simeone, On the complexity of the maximum satisfiability problem for horn formulas, Information Processing Letters 26 (1) (1987) 1–4.
[21] K. Kersting, L. De Raedt, Bayesian logic programming: Theory and tool, in: An Introduction to Statistical Relational Learning, MIT Press, 2007.
[22] J.R. Kok, M.T.J. Spaan, N. Vlassis, Multi-robot decision making using coordination graphs, in: Proc. of the Eleventh ICAR (ICAR'03), 2003.
[23] H.B. Mitchell, Multi-Sensor Fusion: An Introduction, Springer, 2007.
[24] M. Montemerlo, S. Thrun, H. Dahlkamp, D. Stavens, S. Strohband, Winning the DARPA grand challenge with an AI robot, in: Proc. of the Twenty-First AAAI (AAAI'06), 2006.
[25] A. Pfeffer, D. Koller, B. Milch, K.T. Takusagawa, SPOOK: A system for probabilistic object-oriented knowledge representation, in: Proc. of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99), 1999, pp. 541–550.
[26] G. Pinkas, Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge, Artificial Intelligence 77 (1995) 203–247.
[27] D. Poole, The independent choice logic for modeling multiple agents under uncertainty, Artificial Intelligence 94 (1–2) (1997) 7–56.
[28] D. Poole, First-order probabilistic inference, in: Proc. of the Eighteenth IJCAI (IJCAI'03), 2003, pp. 985–991.
[29] P. Pu, B. Faltings, M. Torrens, User-involved preference elicitation, in: IJCAI'03 Workshop on Configuration, 2003.
[30] M. Puterman, Markov Decision Processes, Wiley, New York, 1994.
[31] M. Richardson, P. Domingos, Markov logic networks, Machine Learning 62 (2006) 107–136.
[32] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
[33] T. Sandholm, C. Boutilier, Preference elicitation in combinatorial auctions, in: Y. Shoham, P. Cramton, R. Steinberg (Eds.), Combinatorial Auctions, MIT Press, 2006.
[34] S. Sanner, C. Boutilier, Practical solution techniques for first-order MDPs, Artificial Intelligence 173 (5–6) (2009) 748–788.
[35] Y. Shechter, Control of cooperative mas using preference rules, Master's thesis, Department of Computer Science, Ben-Gurion University, 2010, in press.
[36] S.E. Shimony, Finding MAPs for belief networks is NP-hard, Artificial Intelligence 68 (2) (August 1994) 399–410.
[37] J. Uckelman, Y. Chevaleyre, U. Endriss, J. Lang, Representing utility functions via weighted goals, Mathematical Logic Quarterly 55 (4) (2009) 341–361.
[38] B. Van Roy, J.N. Tsitsiklis, Stable linear approximations to dynamic programming for stochastic control problems with local transitions, in: Proc. NIPS (NIPS'05), 1995, pp. 1045–1051.
[39] C. Wang, S. Joshi, R. Khardon, First order decision diagrams for relational MDPs, Journal of Artificial Intelligence Research 31 (1) (2008) 431–472.