Relational Preference Rules for Control

Ronen I. Brafman
Department of Computer Science
Ben-Gurion University
brafman@cs.bgu.ac.il

Abstract

Much like relational probabilistic models, the need for relational preference models arises naturally in real-world applications where the set of object classes is fixed, but object instances vary from one application to another as well as within the run-time of a single application. To address this problem, we suggest a rule-based preference specification language. 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 it induces a generalized-additive value function. Throughout the paper we use the example of a decision support system for command and control centers we are currently developing to motivate the need for such models and to illustrate them.

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 (Pu, Faltings, & Torrens, 2003; Chen & Pu, 2007; Brafman, Domshlak, & Kogan, 2004; Blythe, 2002), tools for preference elicitation in combinatorial auctions (Sandholm & Boutilier, 2006), recommender systems (Chen & Pu, 2007), 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., (Braziunas & Boutilier, 2005; Gonzales & Perny, 2004)). 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 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 adapt 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 formalism must provide sufficient expressiveness while remaining intuitive.

The general idea of decision-theoretic design is not a new one (for instance, it pervades (Russell & Norvig, 1995)). Recent work in robotics shows that this idea can be very successful (Montemerlo, Thrun, Dahlkamp, Stavens, & Strohband, 2006). Taking it to an extreme implies building agents that maintain beliefs about their states via some probability distribution and goals using a utility function. On the probabilistic side, relational and object-oriented probabilistic models provide tools that allow designers to describe a generic probabilistic model 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 than that of modeling uncertainty. There are many settings in which modeling uncertainty is of lesser concern because designers have a good idea of what they want done as a function of the system data, rather than the real-world state. Moreover, it is often harder to introspect about preferences, and/or to automatically learn them from data. This is especially true when we consider the preferences of a designer. There, unlike in e-commerce applications, it is difficult to see how we can learn from experience

with other "users."

These needs 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 motivate 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 and how systems supporting probabilistic relational models can be used to compute optimal choices. This also provides a first step towards a unified decision-theoretic modeling framework. In addition, we describe a concrete application domain, which is of independent interest, in which we employ this formalism and which serves to motivate it farther.

Preference rules 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. They can be viewed as a special case of soft CLP (Bistarelli, 2004), but with simpler syntax and semantics based on the notion of a generalized-additive value functions (Fishburn, 1970; Bacchus & Grove, 1995).

Like rule-based systems, preference-rule-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.

Like CP-nets, they specify the relative value of some property as a function of a (hopefully) small number of other properties that influence this property. Unlike CP-nets, they are quantitative to allow modularity via aggregation of values using summation, and they use a limited-form of first-order logic to handle a variable number of objects and their attributes. The resulting formalism generalizes classical preference formalisms, and in particular, generalized additive value functions, to domains in which the set of objects vary across time.

In the next section we motivate the use of preference rules using an important application problem: decision support for command and control centers. A system currently under construction which uses preference rules to guide the choice of data displayed to a decision maker in such a context is described in `www.cs.bgu.ac.il/~giorasch/documents.htm`. In Section 3 we describe the syntax and semantics of preference rules. Section 4 discusses some aspects of this formalism. We conclude in Section 5.

Motivation: Command and Control Display

Preference rules arose out of our desire to solve a concrete problem for which we found current techniques inadequate. 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 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, running 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 (Mitchell, 2007), 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.[1]

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 ex-

[1] 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.

ample, 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.

Preference Rules

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 let us capture binary relations between objects.) In addition, we may allow for class attributes. Attributes are separated into two classes: *controllable* and *uncontrollable*. It is possible to enrich this structure with n-ary relations. These introduce complexity, but can also be captured indirectly via set-valued attributes, or multiple binary relationships.

Example 1. Consider a model of the *fire-fighters* domain. 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, fireman 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 some 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 generalized-additive (GA) value function over the set of attribute value-assignments (Fishburn, 1970; Bacchus & Grove, 1995). Technically, we specify a function that, given a set of objects returns a GA value function over the attribute values of these objects. This is done by using *preference rules* with the following form:

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

Where *rule-body* has the following form:

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

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. 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. [2]

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:

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

Note that this will have the same effect as

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

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

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

However, despite its naturalness, 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:

$$(2)\ \text{fireman}(x) \wedge x.co_2\text{-level} = \text{high} \rightarrow x.\text{oxygen-level-display} : \langle \text{(“on”, 10),(“off”,0)} \rangle.$$

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 lifetime 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.

Preference rules allow this flexibility because they are basically schemas of ground rules, and the concrete set of ground rules depends on identity of the objects. A set of objects induces a set ground rule. A ground rule instance is

[2] We use an object-oriented notation which we find more natural for these types of models than the more traditional relational one.

obtained by assigning a concrete object from the appropriate class to the rule variables. Thus, given a rule

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

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:

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

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

Example 3. Consider Rule (1). 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, Rule 2 has a single ground instance with an empty body:

$$\text{Alice.}co_2\text{-level} = \text{high} \rightarrow \text{Alice.oxygen-level-display} : \langle(\text{“on”},10),(\text{“off”},\text{-}10)\rangle$$

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

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

If we add another fireman, Bob, then Rules (1) and (2) 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 $\mathcal{A} = \{A_1, \ldots, A_m\}$ of attribute instances with respective domains D_i. These are precisely the attributes associated with the objects in $\mathcal{O}$. Naturally, if objects are added or removed, then $\mathcal{A}$ 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) must have a value in $\mathcal{O}$. This bounds the size and number of possible interpretations.[3]

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 $\mathcal{A}$. 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, Let r be a ground rule of the form $\alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha : \langle(v_1, w_1), \ldots, (v_k, w_k)\rangle$. Let $\bar{a}$ be a possible assignment to the current objects' attributes. We say that r is *satisfied* by $\bar{a}$ if $\alpha_1, \ldots, \alpha_m$ are satisfied given the standard semantics of the "." operator and the relational operators when the objects' attribute values are given by $\bar{a}$.

The contribution to the value function of r given assignment $\bar{a}$ is 0 either if r is not satisfied by $\bar{a}$ or if r is satisfied by $\bar{a}$ but the value of the head of r according to $\bar{a}$ does not appear as one of the values in the list $\langle(value_1, weight_1), \ldots, (value_k, weight_k)\rangle$. Otherwise, it is the weight w_i associated with the value of the head.

We can now define the value function $v_{\mathcal{R},\mathcal{O}}$ induced by a rule-base $\mathcal{R}$ on a set of objects $\mathcal{O}$.

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

where $v(\bar{a})$ denotes the value $\bar{a}$ assigns to the head of rule r. When $\mathcal{R}$ and/or $\mathcal{O}$ are fixed by the context, we shall omit these subscripts.

Example 4. 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:

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

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

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 on the fire location and we display his camera, the value is 4. If we have two firemen on location then the value is 4 if we display a single camera, and 8 if we display both. Under all other assignments, the value if 0.

One semantic issue is worth considering at this point. Imagine that we add the rule:

$$(3)\ \text{fireman}(x) \wedge \text{fireman}(y) \wedge \text{fire}(z) \wedge x \neq y \wedge x.\text{location} = y.\text{location} \wedge x.\text{location} = z.\text{location} \wedge x.\text{camera.display} = \text{“on”} \rightarrow y.\text{camera.display} : \langle(\text{“on”},\text{-}4)\rangle$$

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 (3), 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). Should we allow this? 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 relational probabilistic models. Bayesian Logic (Kerting & Raedt, 2007), for instance, allows multiple rules with the same head, while Relational Bayesian Networks (Jaeger, 1997), for instance, requires unique rule-heads.

We have opted for the first path, i.e., we allow multiple applicable rules. There are two good reasons for this. First, the aggregation of multiple rules is very simple semantically — we sum up their values, making the semantics of multiple rules much more transparent. This is not the case with probabilistic rules because different aggregation operators makes sense in different settings. Moreover, the second approach is simply a special case of the first, and whatever algorithms work for the first will work for the second.

[3] This assumption can be slightly relaxed.

More importantly, we found it easier to express certain natural preferences that arose in our application domain using this choice. Consider the following preference: "displaying a second camera from the same location does *not* have an added value, although displaying *a* camera *does* have a value." Neither Rule (1) nor Rule (3) alone capture this preference. Rule (3) would discourage us from displaying multiple cameras, but would not encourage having at least one camera. Together, they achieve the desired result: we do value the display of a camera, as expressed by Rule (1), and we do want to penalize the display of redundant cameras, as expressed by Rule (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 $\mathcal{A}$, respectively, and let C denote the set of constraints. Let $\mathbf{u}, \mathbf{c}$ denote an assignment to these attributes. Our goal is to compute: $\max_{\mathbf{c}:\ \mathbf{c} \text{ satisfies } C} v_{\mathcal{R},\mathcal{O}}(\mathbf{u}, \mathbf{c})$, 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.

Here we propose three methods and consider their relative advantage, albeit without an empirical evaluation. The first method is based on local search. It is not guaranteed to return an optimal assignment, but has an anytime property which is important for our real-time application domain. In addition, it naturally lets us bias attribute values to their current value, which is also desirable in our case. The second method is based on branch & bound search, it guarantees optimality, if run to completion. Intermediate leaf nodes can also be used to return an assignment given time constraints. Both methods use a filtered grounded set of rules — as is common with algorithms for probabilistic relational models (e.g., (Getoor & Tasker, 2007)) — and both are based on well-known general techniques. Finally, the third method is based on reduction to relational probabilistic rules, which will hopefully allow us in the future to integrate utilities with probabilities in a uniform framework.

Rule filtering generates the set of applicable ground rule instances. This is done as follows. Let r be a rule of the form

$$\text{class}_1(x_1) \wedge \ldots \wedge \text{class}_k(x_k) \wedge \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha : (v_1, w_1), \ldots, (v_k, w_k).$$

Let $\mathcal{O}$ be the current set of objects with their uncontrollable values fixed.
(1) The rule filter 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 uncontrollable attributes by its value given the current assignment.
(3) It evaluates any condition α_i that depends only on the value of uncontrolabe attributes.
(4) If a condition evaluates to *false*, this grounding is ruled out.
(5) Conditions that evaluate to *true* are dropped.

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

$$\psi_1 \wedge \ldots \wedge \psi_l \rightarrow \psi(v_1, w_1), \ldots, (v_k, w_k)$$

where each ψ_i has the form $o_i.path_i = r_i$ or $o_i.path_i = o_j.path_j$, where $o_i \in \mathcal{O}$ and $o_i.path_i$ denote an attribute path starting at o_i ending in some controllable attribute, and r_i is a value of some simple type. ψ has the form $o.path = r$, and w_i is the weight associated with value v_i, which is identical to the weight of v_i in the original rule.

Local Search Method

Local search algorithms (Hoos & Stutzle, 2004) start with an initial assignment. At each point, they evaluate alternative "neighboring" assignments, and select the best one. Stochastic local search methods introduce some noise into the process, and will occasionally select a neighbor that is not the best one in order to escape from local maxima. The existence of an auxiliary constraint can complicate things a bit because not every assignment we encounter is legal. The method still works, but now we may lose the anytime property because we cannot bound the number of moves required to find a solution. How serious this problem is will depend on the nature of the constraints. For example, in our application domain the constraint is merely a resource constraint, and ensuring that only feasible assignments are selected is easy. We believe that when the constraints are complicated, the systematic B&B approach may be more useful, but this must be verified empirically.

We now explain our implementation of the local search method. Let A be a controllable attribute, and D_A be its domain. We start by associating a value with every element of D_A. The value of $d \in D_A$ is the maximal weight associated with any rule in which the head is d. If no such rule exists, d is assigned a value of 0. Next, we greedily seek a set of attribute values that satisfy the constraints. In the typical case of a capacity constraint (such as screen area in our application), we simply insert the top values until maximal capacity is reached. The resulting assignment is used to initialize the search.

As neighbors of an assignment we select assignments that differ in the value of a single attribute. In the case of capacity constraints, if after changing the value of an attribute we violate the capacity constraint, we consider all minimal attribute-value changes that restore it. This can be summarized as follows: if an assignment satisfies the constraints, its neighbors are all assignments that differ with it on one attribute value. If an assignment does not satisfy the constraint its neighbors are all assignments that differ with it on one attribute value and contribute to the satisfaction of the constraint, i.e., for capacity constraint — reduce the capacity.

Branch & Bound

Branch & bound (Russell & Norvig, 1995) is a classical search strategy for optimization problems. A tree of partial assignments is traversed while maintaining bounds on the solution quality. These bounds can be used to prune branches that lead to provably sub-optimal solutions. The tree is defined as follows: the root node is an empty assignment. The children of a node n are precisely all assignments that extend the assignment of n by an assignment to the same attribute. The leaf nodes of this tree correspond to all possible complete assignments. The order of the attributes assigned along a branch and the order of the children associated with different values of an attribute have an important effect on the efficiency of the search but not its correctness.

This tree is traversed in some form, usually depth-first or breadth-first. To make this traversal more efficient, bounds are used to prune branches as follows. At each point, we have a lower bound l on the quality of the optimal solution. When a node n is reached, we compute an upper bound $u(n)$ on the value of solutions that extend it (i.e., descendent leaf nodes), and if $u(n) < l$ we know that none of its descendants correspond to optimal assignments, and thus this node can be pruned. If we have auxiliary constraints, we also prune partial assignments that violate the constraints.

Typically, bound generation methods depend on the specifics of the problem. In our case, the nature of the auxiliary constraints, if any, is an important consideration. In our application domain our constraints are on the capacity. Each controllable attribute value has an associated resource cost. There is one distinguished value with cost 0 (essentially a noop). For a lower bound we use the value of the best solution seen so far. The upper bound on a node's value is computed as follows: We associate a value with each attribute-value using the same technique described earlier. The only difference is that rules whose body conflicts with the partial assignment associated with the current node are ignored. Next, on top of attribute values already set by the partial assignment, we collect attribute values in descending order based on value-to-capacity ratio. We stop when the choice of the next attribute value violates the capacity constraint. At this point, we add to the accumulated value the value-to-capacity ratio of the just rejected attribute *times* the remaining capacity. This constitutes an upper bound.

Reduction to Relational Probabilistic Models

We noted earlier some important similarities between the suggested formalism and relational probabilistic models. It is well known that various methods for answering probabilistic queries can be used to answer queries in influence diagrams(e.g., (Zhang, 1998)). Indeed, mathematically, one can normalize any value function so that the resulting function has the form of a probability distribution. Here we show a reduction from preference rules to Markov Logic (Richardson & Domingos, 2006) such that finding an optimal assignment in the original set of preference rules can be done by computing the most-probable explanation of the Markov Logic network.

In Markov Logic, first-order formulas are associated with weights. A Markov Logic theory combined with a set of objects induces a probability distribution of the possible interpretations of the resulting Herbrand Base. The probability of each assignment is proportional to $exp(W)$ where W is the sum of weights associated with each satisfied ground instance of a rule.

Suppose we are given a preference rule of the form

$$b \rightarrow h\langle (v_1, w_1), \ldots, (v_k, w_k)\rangle.$$

We can transform it as follows into a set of weighted formulas, as used in Markov Logic Theories: $\{b \wedge h = v_i : w_i\}$. For example:

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

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. Because argmaxing with respect to W and w.r.t. $exp(W)$ is identical, the result is also the assignment with highest value w.r.t. the set of preference rules. Transformations to other probabilistic relational models exist.

An important benefit of this reduction is that it implies that combining relational models of uncertainty and value is likely to be doable. Apparently, all we will need is to instantiate the observed variables and compute the maximum *a-posteriori* (MAP) assignment with respect to the controllable variables. Tools such as Alchemy (`alchemy.cs.washington.edu`) support such queries.

Disucssion

The idea of using first-order rules goes back a long way. More recently, some probabilistic relational models have employed this syntax. Our preference rules formalism was inspired by these relational probabilistic models. Indeed, there is much similarity between utility functions which capture preferences and probability functions. In fact, probability distributions have a natural multiplicative decomposition (via the chain rule) and thus log probability has an additive form which is the typical form using which value and utility functions are decomposed. And because $\log$ and $\exp$ are monotonic functions, maximizing with respect to p and $\log(p)$ leads to the same result. 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, many of the core issues that appear in

relational probabilistic models come up in the case of relational preference models. For example, earlier, we discussed the question of combining rules with identical heads. Many of the computational questions are similar. Indeed, the standard approach in the context of probabilistic models is to ground the first-order rules into propositional rules and use a known techniques for the model obtained (e.g., Bayesian network, Markov network, etc.), much like our approach of generating a grounded GA value function.

Relational preference rules are also closely related to soft constraint logic programs (see (Bistarelli, 2004) for an overview.). The model theoretic semantics of SCLP is similar to that of CLP, i.e., in terms of an interpretation over the Herbrand-Base. But where as in CLP atoms are mapped into truth values, in SCLP atoms are mapped into a possibly richer domain, and the truth value of logical sentences is derived from that of the atoms using appropriate composition operators. The relevant choice in our case is one which maps atoms into reals, and addition is used to assign value to conjunction. Semantically, the main differences are that our models are value functions, which map possible assignments of truth values to controllable variables into the reals; thus, our model theory is not concerned with the value of arbitrary logical expressions; and the value assigned to identical heads by different rules is summed up. Our syntax, too, is more restrictive, ensuring both simplicity and tractability.

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

Theorem 1. *Computing an optimal assignment to the controllable attributes given a preference-rule base, a set of objects, and an assignment to the uncontrollable attributes is NP-hard.*

Proof. By reduction from the problem of finding the most probable explanation (MPE) to a Bayesian network or a join-tree i.e., an assignment to its variables with maximal probability. MPE is known to be NP-hard (Shimony, 1994). It is well known that this is identical to the problem of finding the most preferred assignment to a generalized-additive value function (the multiplicative factoring of probabilistic models leads to an additive factoring of their logarithm, which can be viewed as a generalized additive value function). It is straightforward to generate a set of rules that will yield a desirable GAI value function for a set of objects.

On the other hand, due to its restricted syntax and the closed-world assumption we make, the problem of finding an optimal assignment is decidable, and in fact, is tractable under certain conditions. Given a set of ground rules and objects, we can bound the complexity as exponential in the tree-width of the resulting cost-network. That is, we create a graph whose nodes correspond to concrete attributes (i.e., attributes of concrete object instances). Two nodes are connected if the attributes associated with them appear in the same rule. The problem of finding an optimal assignment can be solved in time polynomial in the input and exponential in the tree-width of this graph. The number of ground rule instance is potentially exponential in the number of variables within a rule. Thus, overall, the complexity is exponential in the maximal number of variables within a rule, and the tree-width of the induced cost-network.

Summary and Future Work

We presented a new flexible approach to preference specification based on preference rules and algorithms for selecting optimal choices given such rules. Preference rules are a special case of Soft CLP, and have numerous 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.

We are currently implementing a command & control based monitoring system as discussed in Section 2. Because we do not have the resources to implement it on a real C&C system, we are emulating one using the "Capture the Flag" mode of the multi-player shooter game *Unreal Tournament*. By recording game playing sessions, we obtained video streams representing different cameras. We use the preference-rules formalism to describe the value of different input streams under different conditions. We are now implementing the first two optimization algorithms described earlier and we will examine their performance. Following that we plan to conduct user studies to evaluate the effectiveness of the selection algorithm. Project progress and documents are available at `www.cs.bgu.ac.il/~giorasch/documents.htm`.

This work provides a rich set of questions for future work. A key issue is the integration of relational preference and probability models, to which our reduction from preference rules to Markov Logic provides an initial step. A firmer foundation for a theory of utility is also desirable, as well as a more useful complexity analysis, i.e., one that can bound the complexity as a function of the original rules rather than the ground rules. Finally, very interesting is the problem of learning preference rules. The latter we anticipate would require algorithms different from those used for learning probabilistic models because of the different input one expects in each context, i.e., in our case, user choices are the outcome of an optimization process.

Acknowledgements: I am grateful to Manfred Jaeger for his help relating this work to relational probabilistic models, to Liron Himi, Ofer Schonberger, Yuval Shahar, and Giora Shcherbakov, for system development, and to the Paul Ivanier Center for Robotics Research and Production Management, the Lynn and William Frankel Center for Computer Science, and COST action IC0602 for their financial support.

References

Bacchus, F., & Grove, A. (1995). Graphical models for preference and utility. In *Proc. 11th UAI*, pp. 3–10.

Bistarelli, S. (2004). *Semirings for Soct Constraint Solving and Programming*. Springer-Verlag. Chapter 6.

Blythe, J. (2002). Visual exploration and incremental utility elicitation. In *AAAI'02*, pp. 526–532.

Brafman, R., Domshlak, C., & Kogan, T. (2004). Compact value-function representations for qualitative preferences. In *UAI'04*, pp. 51–58.

Braziunas, D., & Boutilier, C. (2005). Local utility elicitation in GAI models. In *UAI'05*, pp. 42–49.

Chen, L., & Pu, P. (2007). Preference-based organization interface: Aiding user critiques in recommender systems. In *Proc. of Int. Con. on User Modeling*.

Fishburn, P. C. (1970). *Utility Theory for Decision Making*. John Wiley & Sons.

Getoor, L., & Tasker, B. (2007). *An Introduction to Statistical Relational Learning*. MIT Press.

Gonzales, C., & Perny, P. (2004). GAI networks for utility elicitation. In *KR*, pp. 224–234.

Hoos, H. H., & Stutzle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.

Jaeger, M. (1997). Relational Bayesian networks. In *UAI'97*, pp. 266–273.

Kerting, K., & Raedt, L. D. (2007). *An Introduction to Statistical Relational Learning*, chap. Bayesian Logic Programming: Theory and Tool. MIT Press.

Mitchell, H. (2007). *Multi-Sensor Fusion: An Introduction*. Springer.

Montemerlo, M., Thrun, S., Dahlkamp, H., Stavens, D., & Strohband, S. (2006). Winning the darpa grand challenge with an ai robot. In *AAAI'06*.

Pu, P., Faltings, B., & Torrens, M. (2003). User-involved preference elicitation. In *IJCAI'03 Workshop on Configuration*.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*, 107–136.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A modern approach*. Prentice Hall.

Sandholm, T., & Boutilier, C. (2006). Preference elicitation in combinatorial auctions. In Cramton, Shoham, & Steinberg (Eds.), *Combinatorial Auctions*. MIT Press.

Shimony, S. E. (1994). Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, *68*(2), 399–410.

Zhang, N. L. (1998). Probabilistic inference in influence diagrams. *Computational Intelligence*, *14*, 475–497.