Lifted Optimization for Relational Preference Rules

Ronen I. Brafman BRAFMAN@CS.BGU.AC.IL
Dept. of Computer Science, Ben-Gurion University, Israel

Yagil Engel YAGILE@IE.TECHNION.AC.IL
Dept. of Industrial Engineering and Management, Technion, Israel

Abstract

We present an optimization method for relational preference rules. Our optimization does not require explicit enumeration of the ground rules and thus avoids exponential dependency on the number of objects.

1. Preference Rules

The move to relational probabilistic models from propositional models stemmed from the realization that the type of knowledge we have and need in many domains is at a level more generic than that of concrete objects. In reasoning about preference, too, often we have knowledge about the desirable behavior or state of a system of agents/objects, that applies to different instantiations of this system, while instantiations may differ in the number and properties of concrete objects.

As an example, imagine the problem of monitoring emergency services in a large city. Our objects are fire-fighters, fire-engines, fire-events, injured civilians, and various devices that help us monitor the state of this system, such as cameras and other mounted and stationary sensors. These instruments transmit huge amounts of data to a control center, and our system must decide which information (e.g., which video streams) to display to a decision-maker monitoring this system at each point in time. As the set of objects and their properties change overtime (e.g., as new fire events occur), the logic behind what is desirable and what is less desirable can (and must) be described at a generic level, so that it will apply to different concrete instances of such a monitoring system.

In (Brafman, 2008) we proposed relational preference rules (RPR) as a formalism for modeling preference/value information about such a system. RPRs are similar in form to existing PRMs, such as Bayesian Logic (Kersting & Raedt, 2007), Relational Bayesian Networks (Jaeger, 1997), and Markov Logic (Richardson & Domingos, 2006), and can be viewed as relational UCP networks (Boutilier et al., 2001), which induce a GAI value function over any given set of objects. We illustrate the basics of this formalism here, and refer the reader to (Brafman, 2008) for more details. In this paper we concentrate on lifted inference for relational preference rules which turns out to be quite different from lifted probabilistic inference.

For the purpose of designing 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 of a simple type, such as integers, reals, strings, or an object class. However, in this paper we adopt a more constrained model in which an attribute cannot refer to an object. Attributes are separated into two classes: *controllable* and *uncontrollable*. The uncontrollable attributes can be viewed as specifying the context in which we operate. However, in this abstract we ignore them for the most part.

We define *preference rules* to have the syntax:

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

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: $x_i.attr\ REL\ value$ or $x_i.attr_i\ REL\ x_j.attr_j$. Each x_i must appear earlier within a $\text{class}_j(x_i)$ element. *REL* denotes a relational operator such as $=, \neq, >, <$ etc.

The *rule-head* has the form $x_j.attr$ where $x_j.attr$ denotes a controllable attribute. $\langle (v_1, w_1), \ldots, (v_k, w_k) \rangle$ is a list of pairs, where v_i denotes a possible value of the attribute in *rule-head*, and w_i is a real-valued weight. For example, the following rule expresses the

Presented at *ILP-MLG-SRL*, Leuven, Belgium, 2009.

fact that viewing the stream generated by a fireman in a location of a fire has value 4:

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

Every set of concrete 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 parameters. Thus, given a rule

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

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

$$\alpha'_1 \wedge \ldots \wedge \alpha'_m \rightarrow \alpha' \ \ \langle(v_1, w_1), \ldots, (v_k, w_k)\rangle$$

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

A given set of preference rules and a concrete set of objects induce a value function over the different possible assignments to the controllable attributes of these objects. The value of any particular assignment is simply the sum of values associated with these objects under this assignment. That is, we sum up all the weights w associated with ground rules r such that: (1) r is a ground instance of some rule induced by this set of objects; (2) the body of r is satisfied by the attribute values of the relevant objects; (3) the value of the attribute at the head of r according to this assignment is v and r associates weight w with the value v.

Our main computational task given a rule-base $\mathcal{R}$ and a set of objects $\mathcal{O}$ with a given assignment to their uncontrollable attributes is to find an assignment to the controllable attributes that is optimal. The straightforward way to do that is by creating all the ground instances of the rules, and evaluating different possible assignments with respect to this set of ground rules. Unfortunately, the optimization domain's size is exponential in the number of objects per class, multiplied by the number of attribute per class. Although some improvements can be made at the level of the ground representation (e.g., the use of variable elimination to solve this maximization problem), this is a prohibitive factor. The motivation for this work is hence to solve this problem without explicit grounded enumeration.

2. Lifted Optimization

In most cases, the value of a complete assignment does not depend on *which* ground object got which assignment for its attributes, but rather on *how many* of them were given each assignment. That is, in the case of an additive model such as ours, what matters is how many times each rule was fired with a certain value. This property can be leveraged to perform the optimization in the lifted (first-order) rule level, without ever creating the explicit set of corresponding grounded rules.

For the purpose of optimization we slightly transform the syntax of the rules, taking it closer to that of Markov Logic. Each rule with a head α is replaced with a set of rules, one for each weighted value v_i of α. The predicate $\alpha = v_i$ is added to the body of each new rule, whose weight is w_i. Each rule $r \in R$ is now a simple conjunction of predicates with one weight w_r.

For simplicity, assume first that there is single class of objects C with a single attribute A with domain $c_1, \ldots, c_d$. Hence, any instance of class C falls into one of d types, distinguished by their attribute value. Let us further assume a single rule r with weight w that contains a single parameter. Thus, whether or not an object of class C causes r to fire depends only on the value of its attribute A. The total weight of a particular assignment of values to N objects of class C depends only on N_j, for $j = 1, \ldots, d$, the number of object instances for which $A = c_j$. Specifically, the value of an assignment would be

$$w \cdot \sum_{j=1}^{d} 1_{A=c_j \text{ causes } r \text{ to fire}} N_j.$$

Given m rules $r_1, \ldots, r_m$ with respective weights $w_1, \ldots, w_m$, and a single parameter each, the value of an assignment would be $\sum_{k=1}^{m} w_k \cdot \sum_{j=1}^{d} 1_{A=c_j \text{ causes } r_k \text{ to fire}} N_j$.

Next, consider a rule r with two parameters, x and x'. Let $SAT(r)$ refer to pairs of values to A, one for x and one for x', that together cause r to fire. Now the number of times the rule will fire is the number of *pairs of object instances* that satisfy this rule:

$$w \cdot \sum_{j=1}^{d} \sum_{k=1}^{d} 1_{(c_j,c_k) \in SAT(r)} N_j \cdot N_k.$$

For example, if the rule fires when the two parameters are assigned objects with the same attribute value, we would get $w \cdot \sum_{j=1}^{d} N_j^2$.

More generally, we can express the value of any assignment in terms of a function that depends on some constants (the rule weights) and the number of object instances from each complete object profile, where by a complete profile we mean a vector of the values assigned to each attribute of the object. In order to find

the optimal assignment we optimize over all possible such counts, under the *capacity constraint* of the class, meaning that the counts per class sum up to the number of objects in the class.

This computation avoids exponential dependence in the number of objects, but requires a number of variables which is exponential in the number of attributes per class (one *counter* for each complete object profile). Therefore, the size of the optimization domain is double exponential in this term. We next show how this complexity can be improved significantly—intuitively, by avoiding the enumeration of assignments that do not satisfy any rule.

2.1. Direct Counting of Satisfying Assignments

Let C^i denote a class, and let $CL_r(x)$ denote the class of parameter x in rule r. For each parameter $r.x$ (indicating parameter x within rule r), we define a counter $n_{r.x}$ which indicates the number of objects of $CL_r(x)$ that satisfy the predicates involving x in r. Assume that all the predicates are of the form $x.A_i = v$, that is no predicate involves multiple parameters (we show in the tech report how to lift this restriction[1]). The number of times r is triggered is now simply $\prod_{x_i} n_{r.x_i}$, where the product is over parameters appearing in r.

In the previous scheme we applied simple capacity constraints to ensure that the size of each class is respected. We cannot use similar constraints here, because the summation over all the counters $n_{r.x}$ of a class need not be bounded by the number of objects in the class. The reason is that the same object can satisfy several rules. For example, assume that the predicates of the three parameters $r.x$, $r'.x'$, and $r''.x''$ are not contradicting, that is they can all be satisfied by a given object. The constraint $n_{r.x} + n_{r'.x'} + n_{r''.x''} = |C^i|$ is too constraining, because for example objects that satisfy both $r.x$ and $r'.x'$ are counted twice.

In order to avoid double counting, we define a counter $n_{r.x,r'.x'}$ that counts the number of objects that satisfy the first two parameters, and similarly for the other two pairs, leading to a constraint $n_{r.x}+n_{r'.x'}+n_{r''.x''}-n_{r.x,r'.x'} - n_{r.x,r''.x''} - n_{r'.x',r''.x''} = |C^i|$. Unfortunately, objects that satisfy all three parameters are now reduced three times, whereas the original count over-counted them only twice. We therefore define a counter for the three-way intersection and add it to the count. This leads to terms formed as the inclusion–exclusion counting form of set theory. We pose such a constraint for each maximal clique of related parameters, that is parameters referring to mutual attributes.

[1] `http://ie.technion.ac.il/~yagile/prefRules.pdf`

We propose to solve the resulting constraint optimization problem using the well-known *variable elimination* technique, which employs a graph whose nodes represent counters, and two counters are connected by an edge if they relate via a rule or via mutual attributes. This graph captures the structure of both the objective function and the constraints. The optimization solution includes a value for each counter, which can be translated efficiently to a complete ground assignment. For details and performance analysis we refer to the technical report.

2.2. Offline Computation

Often, the set of objects in the system changes overtime, and hence this optimization problem has to be solved online; potentially causing a delay that is visible to the user. Fortunately, we can generate an approximation to the optimal assignment using an offline computation, that is independent of the number of objects per class. The idea is that we can replace the explicit counting variables with *ratio* variables, which indicate the portion of class objects that take each assignment. The offline problem is a continuous optimization problem with multilinear objective function and linear constraints. Several algorithms have been introduced for this problem (see De Campos and Cozman (2004) for a survey). While not polynomial, branch and bound methods perform reasonably well if the degree of the product is not too large. In our case the degree is the number of parameters per rule, which we assume to be a small constant. Online, we scale the optimal fractional values by the number of objects in each class and round them. The (scaled) fractional solution's value is an upper bound on the optimal value, hence the difference between the values of the fractional and integer solution bounds the quality of the latter.

References

Boutilier, C., Bacchus, F., & Brafman, R. I. (2001). UCP-networks: A directed graphical representation of conditional utilities. *UAI*.

Brafman, R. I. (2008). Relational preference rules for control. *KR*.

De Campos, C. P., & Cozman, F. G. (2004). Inference in credal networks using multilinear programming. *Starting AI Researcher Symposium*.

Jaeger, M. (1997). Relational Bayesian networks. *UAI*.

Kersting, K., & Raedt, L. D. (2007). *An introduction to statistical relational learning*. MIT Press.

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