Proceedings, Fifteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2016)

Probabilistic Models over Weighted Orderings: Fixed-Parameter Tractable Variable Elimination

Thomas Lukasiewicz
Department of Computer Science
University of Oxford, UK
firstname.lastname@cs.ox.ac.uk

Maria Vanina Martinez
Institute for Computer Science and Engineering
(Universidad Nacional del Sur–CONICET), Argentina
mvm@cs.uns.edu.ar

David Poole
Department of Computer Science
University of British Columbia, Canada
poole@cs.ubc.ca

Gerardo I. Simari
Institute for Computer Science and Engineering
(Universidad Nacional del Sur–CONICET), Argentina
gis@cs.uns.edu.ar

Abstract

Probabilistic models with weighted formulas, known as Markov models or log-linear models, are used in many domains. Recent models of weighted orderings between elements that have been proposed as flexible tools to express preferences under uncertainty, are also potentially useful in applications like planning, temporal reasoning, and user modeling. Their computational properties are very different from those of conventional Markov models; because of the transitivity of the "less than" relation, standard methods that exploit structure of the models, such as variable elimination, are not directly applicable, as there are no conditional independencies between the orderings within connected components. The best known algorithms for general inference in these models are exponential in the number of statements. Here, we present the first algorithms that exploit the available structure. We begin with the special case of models in the form of chains; we present an exact $O(n^3)$ algorithm, where n is the total number of elements. Next, we generalize this technique to models in which the set of statements are comprised of arbitrary sets of atomic weighted preference formulas (while the query and evidence are conjunctions of atomic preference formulas), and the resulting exact algorithm runs in time $O(m * n^2 * n^c)$, where m is the number of preference formulas, n is the number of elements, and c is the maximum number of elements in a linear cut (which depends both on the structure of the model and the order in which the elements are processed)—therefore, this algorithm is tractable for cases in which c can be bounded to a low value. Finally, we report on the results of an empirical evaluation of both algorithms, showing how they scale with reasonably-sized models.

Introduction

Many modern applications of AI run across the problem of reasoning about different ways in which elements can be ordered—common examples include planning (actions must be ordered with respect to execution precedence), temporal reasoning (objects in a domain are ordered relative to a timeline), recommender systems (items are ordered with respect to user preferences), semantic search (query results are ordered with respect to relevance and also potentially user preferences), social choice (where options to choose from in an election are ordered with respect to feedback from a population), and sports (where we might want to order players or teams with respect to ability, or know the likely outcome of a match, which is central to betting), among others.

Since such applications often also must work under uncertainty, it is natural to explore how probabilistic models can be extended to work in cases where, instead of having possible worlds defined over truth values for a set of Boolean variables, there are possible worlds over the (linear) orderings of elements of interest. As in conventional probabilistic models, this would allow us to answer conditional probabilistic queries about *any combination* of elements given any others. Even though tools to represent and reason with probability distributions over Boolean variables have been well studied and are quite mature, distributions over linear orderings have received much less attention; tractable, flexible tools are still lacking. The main hurdle to overcome is thus essentially a KR one—while the former have a local independence structure that can be exploited, the latter have no such local independence (cf. Example 3 below), but still have structure. Applications of representations of such structured spaces can be found in the literature of various areas; see, e.g., (Bozga and Maler 1999) and (Kisa et al. 2014).

In this paper, we continue recent work (Lukasiewicz, Martinez, and Simari 2014), where we proposed a novel Markov model akin to Markov Logic (Richardson and Domingos 2006) in which weighted formulas express preferences among different elements. Though the knowledge expressed in such models may also be expressed in other formalisms (e.g., probabilistic logic programs (Raedt, Kimmig, and Toivonen 2007), with the addition of transitivity and other axioms, which makes them impractical), this language is especially designed for reasoning about probability distributions over linear orderings based on incomplete and uncertain information. The complexity of the algorithms proposed in (Lukasiewicz, Martinez, and Simari 2014) for inference in weighted preference models involves exponential factors in the *number of statements* in the model; here, we show how we can exploit the structure of the models to do exact inference much more efficiently when the model is comprised of atomic formulas and the query/evidence are conjunctions of atomic preference formulas. Even though the worst case complexity of our algorithms is exponential, the dominating factor is the *linear cut size* of the model (a measure similar to treewidth, as explained below); on the other hand, our previous algorithms are exponential in the *number of statements* in the model. Thus, even when the linear cut size can be bounded, for connected models the number of statements depends directly on the number of elements, so the algorithms from (Lukasiewicz, Martinez, and Simari 2014) cannot scale in the same way.

The following example serves as a simple introduction to Markov models over weighted orderings, applied to the domain of reasoning with preferences for user modeling (Chomicki 2003; Rossi, Venable, and Walsh 2011).

Example 1. Suppose we receive the following information from a user of an online movie rating system regarding movie category preference: (i) they very likely prefer comedies over dramas; (ii) they generally prefer comedies over action films; and (iii) they almost always prefer action films over horror movies. Such information may come in the form of quantitative assessments obtained by automated analysis of user activity, directly from the user in the form of weights or probabilities, or a combination of the two.

We may want to know how likely it is for this user to prefer comedies over horror films; note that this information is stated only implicitly. ■

We refer the reader to the related work section for a discussion on existing models for this kind of reasoning and their relation to the one adopted here.

Preliminaries

We now discuss necessary preliminary concepts on conventional Markov models and weighted preference formulas.

Markov Random Fields

The semantics of Markov models over weighted preference statements is defined in terms of classical models known as Markov random fields (MRFs), or Markov networks (Koller and Friedman 2009), which are probabilistic models that represent a joint probability distribution over a (finite) set of random variables $X = \{X_1, \ldots, X_n\}$. Each X_i may take on *values* from a finite *domain* $Dom(X_i)$. A *value* for $X = \{X_1, \ldots, X_n\}$ is a mapping $x\colon X \to \bigcup_{i=1}^{n} Dom(X_i)$ such that $x(X_i) \in Dom(X_i)$; the *domain* of X, denoted $Dom(X)$, is the set of all values for X. Thus, a value for X consists of an assignment of a value to each X_i. An MRF is defined in terms of a set of potential functions ϕ_j, where each is a function from a subset of the variables into the non-negative real numbers. The probability of a value for X is a product of the values for the potentials for the assignment. An MRF induces an undirected graph, where the nodes are random variables and there is an arc between every pair of nodes that appear together in a potential. For positive MRFs (i.e., no zeros in the potentials), the graph induces a dependency relation such that a variable is independent of its non-neighbors given its neighbors.

For positive MRFs, there is an equivalent *log-linear* representation, which involves defining a set of *features* for the potentials; a feature is a real-valued function of the state of a set of variables. Given a value $x \in Dom(X)$ and features f_j, the probability distribution represented by a log-linear MRF is given by $P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j \cdot f_j(x)\right)$, where j ranges over the set of features, and $w_j = \log \phi_j(x_{\{j\}})$ (here, $x_{\{j\}}$ is the projection of x onto the domain of the j-th feature). Term Z is a normalization constant to ensure that $\sum_x P(X = x) = 1$. Thus, $Z = \sum_{x \in Dom(X)} \exp\left(\sum_j w_j \cdot f_j(x)\right)$.

Probabilistic inference in MRFs is intractable (Roth 1996). For sparse graphs, exact inference (Koller and Friedman 2009; Darwiche 2009) can be carried out in time exponential in treewidth, using methods such as variable elimination or clique trees. Approximate inference mechanisms, such as Markov Chain Monte Carlo, have also been developed and successfully applied in practice.

Weighted Preference Statements

Let $\mathscr{U}$ be the universe of *elements*, with $|\mathscr{U}| = n$—these are the things being compared. Given $a, b \in \mathscr{U}$, an *atomic preference statement* is of the form $a \succ b$ and denotes that element a is preferred to element b. Even though the formalism of (Lukasiewicz, Martinez, and Simari 2014) allows more general statements (i.e., conjunctions, disjunctions, and negations), the algorithms in this paper deal with the case in which all statements are atomic; therefore, we often refer to atomic preference statements simply as preference statements.

Possible worlds are defined in terms of permutations (linear orderings) of the elements of $\mathscr{U}$; worlds are thus injective (1 to 1) functions from $[1..|\mathscr{U}|]$ to $\mathscr{U}$. There are a total of $n!$ possible worlds, and we denote the set of all worlds with Ω. A world ω is said to *satisfy* a statement $a \succ b$, denoted $w \models a \succ b$, iff a appears before b in the permutation.

A *weighted (preference) formula* is a pair $\langle f, w \rangle$ where f is a preference statement over elements of $\mathscr{U}$, and w is a non-negative real number; w is said to be the *weight* of formula f. We sometimes write $\langle f, w \rangle$ as $f : w$, and call a set of weighted preference formulas a *model*. The following example presents a model that we will use later in the paper.

Example 2. Consider the following model:

$$f_1 = a \succ b : w_1, \quad f_2 = b \succ c : w_2, \quad f_3 = c \succ d : w_3,$$
$$f_4 = c \succ e : w_4, \quad f_5 = e \succ d : w_5.$$

We use abstract names for convenience, but clearly these formulas can represent the kind of preferences in Example 1. ■

Semantics. In the following, we assume that there is an arbitrary (but fixed) order over the elements in $\mathscr{U}$, denoted by the symbol "$<$". The probabilistic semantics of a model M is given by the Markov random field that is defined as follows (Lukasiewicz, Martinez, and Simari 2014):

(1) for each $a, b \in \mathscr{U}$ with $a < b$, there exists a (binary) node (a, b); the node's value is 1, if $a \succ b$ is true, and 0, otherwise;

(2) one feature is defined for each statement in M, with value 1 iff the corresponding statement is true (and 0, otherwise); the weight of this feature is the weight associated with the statement in M; and

(3) *zero weight* features defined according to the following templates. For each $X,Y,Z \in \mathscr{U}$ such that $X < Y < Z$:

$$\neg\big(((X,Y)=1) \land ((Y,Z)=1) \Rightarrow (X,Z)=1\big);$$
$$\neg\big(((X,Y)=1) \land ((X,Z)=0) \Rightarrow (Y,Z)=0\big);$$
$$\neg\big(((X,Z)=0) \land ((Y,Z)=1) \Rightarrow (X,Y)=0\big);$$
$$\neg\big(((X,Y)=0) \land ((Y,Z)=0) \Rightarrow (X,Z)=0\big).$$

Recall that MRFs do not contain logical variables—these features are described in this schematic way to show how they are built. These features impose a probability of zero on assignments of values to random variables that cause transitivity to be violated.

The MRF thus contains $\binom{|\mathscr{U}|}{2}$ nodes, and an edge between nodes iff the preference atoms associated with such nodes appear together in a feature. Condition 3 establishes a 1-to-1 mapping between linear orderings and value assignments to random variables in the MRF with non-zero probability.

Example 3. Consider the following simple model:

$$M = \{\langle a \succ b, w_b\rangle, \langle b \succ c, w_c\rangle, \langle c \succ d, w_d\rangle\}.$$

So, we have $\mathscr{U} = \{a,b,c,d\}$, and suppose the arbitrary ordering of the elements is $a < b < c < d$. The model has 4 elements and thus $4! = 24$ worlds (linear extensions); the underlying MRF—which provides the probabilistic semantics for the model—is shown in Figure 1. This (conventional) MRF contains $\binom{4}{2} = 6$ Boolean variables (one for each ordered pair of elements X,Y in $\mathscr{U}$ such that $X < Y$).

The following table illustrates (part of) the mapping between the values of the variables in the underlying MRF and the worlds for M; out of the $2^{\binom{4}{2}} = 64$ possible assignments of truth values to Boolean variables, only $4! = 24$ correspond to linear extensions (the rest of them violate the transitivity constraints).

(a,b)	(a,c)	(a,d)	(b,c)	(b,d)	(c,d)	World
1	1	1	1	1	1	$\langle a,b,c,d\rangle$
1	1	1	1	1	0	$\langle a,b,d,c\rangle$
1	1	1	1	0	1	—
1	1	1	1	0	0	$\langle a,d,b,c\rangle$
1	1	1	0	1	1	$\langle a,c,b,d\rangle$
1	1	1	0	1	0	—
1	1	1	0	0	1	$\langle a,c,d,b\rangle$
1	1	1	0	0	0	$\langle a,d,c,b\rangle$
1	1	0	1	1	1	—
1	1	0	1	1	0	—
(54 more lines...)						

■

The *weight* of a world is the product of the weights of the formulas that are satisfied by the world. The *partition function* for model M is:

$$Z(M) = \sum_{\omega\in\Omega} \prod_{\langle f,w\rangle\in M, \omega\models f} w.$$

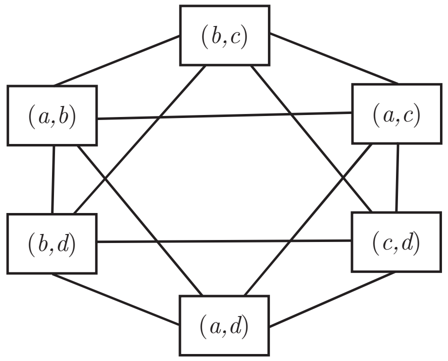

Figure 1: Underlying MRF for the model in Example 3.

The *probability* of a formula given the model is proportional to the sum of the weights of the worlds that satisfy the formula. The probability of any formula α (formed by combinations of atomic statements with $\land$, $\lor$, and $\neg$) is given by:

$$P_M(\alpha) = \frac{1}{Z(M)} \sum_{\omega\models\alpha} \prod_{\langle f,w\rangle\in M, \omega\models f} w.$$

The conditional probability of α given another (evidence) formula e is defined as usual:

$$P_M(\alpha \mid e) = P_M(\alpha \land e)/P_M(e).$$

Note that since weights are multiplied, a weight of 1 is equivalent to not having the formula at all. When M can be understood from the context, it is omitted from the subscript.

As a quick illustration, consider the model from Example 3 and suppose we want to compute $P(a \succ c)$. To compute Z, we could sum 24 terms (one per world), where each term is a product of 3 factors (one per statement), which can be either 1, if the statement is not satisfied by the world, or the weight corresponding to the statement, if it is. For instance, world $w = \langle b,a,c,d\rangle$ gives rise to term $1 \cdot w_c \cdot w_d$, since $w \not\models a \succ b$ (b appears after a in w), $w \models b \succ c$, (since b appears before c), and $w \models c \succ d$ (c precedes d). The numerator can then be computed in the same way, except that we only have terms corresponding to worlds in which a appears before c. Conditional probabilities are completely analogous to their traditional counterpart; e.g., if we have evidence "$c \succ d$", we are only interested in the universe of worlds in which this is true, and we compute the query's probability relative to that universe. Note that these computations correspond to naïvely following the semantics, and that the algorithms that we develop are much more efficient.

Proposition 4. *If α is a conjunction of preference statements $(a_1 \succ b_1) \land \cdots \land (a_k \succ b_k)$, then:*

$$P_M(\alpha) = \frac{Z(M \cup \{\langle b_1 \succ a_1 : 0\rangle, \ldots, \langle b_k \succ a_k : 0\rangle\})}{Z(M)}.$$

Proof. Direct consequence of the definitions of $Z(M)$ and $P_M(\alpha)$; the statements added to the model with weight zero cause the weights of worlds that do not satisfy α to be multiplied by zero, so the non-zero terms remaining in the summation are precisely those in the numerator of $P_M(\alpha)$. □

Thus, to compute arbitrary conditional probabilities of conjunctions, we only need to compute partition functions. As (conditional) probabilities of arbitrary propositions can be computed from conjunctions, for the rest of the paper, we only concentrate on computing the partition function.

It is also possible to work in the log-domain, as in log-linear models, where the weights are arbitrary real numbers that are added. The probability of a world would then be proportional to the exponent of the sum of the weights. This characterization is less general, as it does not allow for zero probabilities (unless it allows for weights of $-\infty$). It is easy to translate the results in this paper into the log-domain.

Conditional (Non-)Independence. Consider again the setup in Example 3 and Figure 1; note that the links between variables in the underlying MRF are dense; in particular, due to the effect of the features enforcing transitivity (Condition 3 in the construction), in every connected component, every element in M is connected to every other variable in the underlying MRF. This can be easily seen in Figure 1, where element a in M appears in (a,b) in the MRF, which is linked directly with (a,c) and (a,d) (among others, but these suffice to cover the remaining elements in M); if M had more elements, say e and f, then (a,b) would be directly linked to (a,e) and (a,f) as well.

As a consequence, even in simple "chain-like" models of the form $\{\langle x_1 \succ x_2, w_2\rangle, \langle x_2 \succ x_3, w_3\rangle, \ldots, \langle x_{n-1} \succ x_n, w_n\rangle\}$, if e is any set of observations regarding the truth of any subset of weighted formulas $x_{i-1} \succ x_i$ (for $3 \leq i \leq n-1$), including the empty set, then $P(x_1 \succ x_2 \mid e, x_{n-1} \succ x_n) \neq P(x_1 \succ x_2 \mid e, x_{n-1} \prec x_n)$. This inequality holds almost everywhere; although there are assignments to make this numerically equal, it is very sensitive to slight variations in weights.

In conventional models of weighted formulas, a standard way to compute probabilities is to exploit the structure by using non-serial dynamic programming (Bertelè and Brioschi 1972), variable elimination (Zhang and Poole 1994; Dechter 1996), message passing in clique trees (Lauritzen and Spiegelhalter 1988), or search-based methods like recursive conditioning (Darwiche 2001), which are linear in the number of variables for a bounded treewidth, and are exponential in the treewidth. These methods have been used for database joins, constraint satisfaction problems, probabilistic inference, and optimization. They rely on a notion of independence: some variable V only depends on a few other variables (its neighbors), and the rest of the variables are independent of V given its neighbors—variable V can thus be eliminated and its influence passed to its neighbors. *Given the complexity caused by the transitivity constraints and the very little independence structure (as can be seen in Figure 1), these methods are not directly applicable for weighted orderings.*

We could try to apply the same variable elimination strategy described above for our model, essentially "eliminating" x_n (in the classical variable elimination sense) by creating a weight w'_{n-1}, so that replacing w_n with 1 (effectively removing the constraint), replacing w_{n-1} with w'_{n-1}, and leaving the rest constant so that $P(x_{n-1} \succ x_{n-2})$ is not affected. The main problem with this approach is that, unfortunately, w'_{n-1} depends on *all* of the weights, including w_1, in a complex manner. Since the techniques that we developed are similar in spirit to traditional variable elimination, we refer to them also as variable elimination algorithms; intuitively, we can picture a graph in which each node is an element, and each edge corresponds to an atomic weighted preference statement—eliminating an element in $\mathscr{U}$ is thus analogous to eliminating a random variable in a conventional MRF.

An Algorithm for Chain Models

We begin with the special case of models that we call *chains*; these models are simply sets of statements of the form:

$$\{\langle x_1 \succ x_2, w_2\rangle, \langle x_2 \succ x_3, w_3\rangle, \ldots,$$
$$\langle x_{i-1} \succ x_i, w_i\rangle, \ldots, \langle x_{n-1} \succ x_n, w_n\rangle\};$$

the model in Example 3 is a chain with four elements. We assume there is a single chain, though multiple chains are easily handled since disconnected components are mutually independent and can be computed separately and combined as in any independence model.

The reason why traditional methods are not directly applicable is also the insight needed to devise an efficient algorithm. Weight w_n affects $P(x_1 \succ x_2 \mid e)$ (for some evidence e), because it affects the position of x_{n-1} in the order, which in turn affects the position of x_{n-2} in the order, and so on, eventually affecting the position of x_1. Observing, say, $x_{n-2} \succ x_{n-1}$ does not stop the position of x_{n-1} affecting the position of x_{n-2}. However, if we knew the *position* of x_2 in the ordering of $x_2, \ldots, x_n$, all subsequent elements are irrelevant when computing $P(x_1 \succ x_2 \mid e)$.

So, if we knew the position of x_{n-1} in the ordering of $x_1 \ldots x_{n-1}$ we could determine the effect of eliminating x_n. If we knew the position of x_{n-2} in the ordering of $x_1 \ldots x_{n-2}$, we could determine the effect of eliminating w_{n-1}, w_n, etc.

Counting Tables. In order to support the efficient computation of the effect of variable elimination steps, we define the concept of *counting table* that, for chains, is simply an array $r[\ ]$ of real values initially of size n with all cells set to value 1 . Elements are eliminated from last to first in the chain, and thus each elimination step involves a single statement ($x_{n-1} \succ x_n$ first, then $x_{n-2} \succ x_{n-1}$, and so on) and reduces the size of the counting table by one. The new values are computed based on the old ones; essentially, we must consider the effect of inserting the element being eliminated in all possible remaining positions, and in each case consider the satisfaction of the statement involved—if it is satisfied, the new value must be multiplied by the corresponding weight (if not, we use the old value, which is equivalent to multiplying by 1). Intuitively, a counting table is a dynamic programming structure that contains the partial information necessary to compute the next step.

To better illustrate the process of building the table, let's go into more detail, considering first the initial step and then the operation in general. We begin by eliminating x_n—consider the case where x_{n-1} is in position i, for $1 \leq i \leq n-1$, in the ordering of $x_1 \ldots x_{n-1}$; this means there are $i-1$ elements before x_{n-1} (not including x_{n-1}). The key intuition here is that the effect of w_n depends on *how many* elements are before x_{n-1}, not *which* elements are before x_{n-1} in the ordering. We can compute the effect of $\langle x_{n-1} \succ x_n, w_n \rangle$ for position i analytically. There are i positions where x_n could be placed before x_{n-1} and $n-i$ positions where x_n could be placed after x_{n-1}; thus, the effect of this tuple is $i + (n-i)w_n$. We compute this value for each i, and store it in the counting table $r[1:n-1]$.

For the recursive step, suppose we have already removed elements $x_n, x_{n-1}, \dots, x_{k+1}$, with $k \le n-1$, and we want to remove element x_k; we have a counting table $r[1:k]$ with values for the remaining elements for each position that x_k can take in orderings containing those elements (that is, elements $x_k \dots x_1$). The meaning of the values in r are as follows: if x_k is in position i, $r[i]$ is the (partial) sum of the weights of the orderings that have x_k in position i. We want to remove x_k and produce a counting table r' for x_{k-1}. Suppose x_{k-1} is in position i (out of the elements $x_{k-1}, \dots, x_1$), the value $r'[i]$ is computed as follows: consider each position where x_k can fit into the ordering. If it goes to position j, and $j \le i$, the value is $1 \times r[j]$, and if $j > i$ then the value is $w_k \times r[j]$. Thus, summing over all the possible positions gives:

$$r'[i] = \sum_{j=1}^{k} r[j] \times \begin{cases} 1 & \textit{if} \quad j \le i \\ w_k & \textit{if} \quad j > i. \end{cases} \qquad (1)$$

We compute this for every i where $1 \le i < k$. The array r' becomes the new counting table r for the next iteration (where we will delete x_{k-1}), and so on and so forth.

Once x_2 has been eliminated, r contains a single value—$Z(M)$—the value of the partition function for model M. Figure 2 contains the pseudocode for this algorithm.

Algorithm ComputeZ-Chain can be used to answer queries as long as the resulting models are chains; this means the observations and queries must be adjacent, or M consists of two disjoint chains, and the query is about an inequality between the end of one chain and the start of the other.

Proposition 5. *Given a chain model of size n, Algorithm* ComputeZ-Chain *runs in time $O(n^3)$.*

Proof. The nested `for` loops in lines 2, 4, and 6 are each executed $O(n)$ times, while the rest of the instructions are $O(1)$. The total running time is therefore $O(n^3)$. □

Thus, Propositions 4 and 5 together tell us that queries over chains can be answered in time $O(n^3)$, as long as the model is a chain after adding the query and observations.

Example 6. Consider the following chain model:

$$f_1 = a \succ b : w_1, \quad f_2 = b \succ c : w_2, \quad f_3 = c \succ d : w_3,$$
$$f_4 = d \succ e : w_4, \quad f_5 = e \succ f : w_5.$$

Elements are eliminated in the order: $[f, e, d, c, b, a]$; the first iteration creates array r with values $r[1:6]$ that are all equal to 1 and represent all possible places where element f can be placed in a permutation of size 6.

When eliminating element f, formula f_5 is affected, so we compute a new array $r'[1:5]$ corresponding to the cases where element e is in the first position, the second, and so on in a permutation of size 5. Thus, $r'[1]$ corresponds to a permutation of e and all other remaining elements; however, since we only care about the position of f with respect to e, the rest of the elements remain unknown, and we denote them with the symbol "$-$". Therefore, $r'[1]$ corresponds to $[e----]$, $r'[2]$ to $[-e---]$, and so on. We now must consider all the ways in which f could be inserted in each permutation; for $[e----]$, we have 6 possibilities:

$$[fe----], \quad [ef----], \quad [e-f---],$$
$$[e--f--], \quad [e---f-], \quad [e----f].$$

Each such partial permutation yields a term that is either 1 or w_5, depending on whether or not it satisfies f_5, times the value stored for the current position of element f. For instance, we have $r'[1] = r[1]*1 + r[2]*w_5 + r[3]*w_5 + r[4]*w_5 + r[5]*w_5 + r[6]*w_5 = 1 + 5*w_5$.

Array r' replaces array r; after computing this array, we remove formula f_5 from consideration and continue with the elimination of element e. This elimination affects formula f_4, so we therefore compute a new array $r'[1:4]$ corresponding to the cases where element d is in the first position, the second, and so on in a permutation of size 4. This process continues until we have eliminated all elements, in which case we have an array with a single cell containing the value of the partition function Z.

Consider the query $Q : P(c \succ d)$; to compute this value, we add the weighted preference statement $\langle d \succ c, 0 \rangle$. Now, when the process concludes, the single-cell array contains the value Z' such that $P(c \succ d) = Z'/Z$. ■

```
Algorithm ComputeZ-Chain($M$)
Input: Weighted preference model $M$ over $n$ elements.
Output: The value of the partition function for $M$.
 1. Initialize array $r[1:n]$ to $1^n$;
 2. for $i$:= $n$ downto 2 do
 3.    let $f_i = (x_{i-1} \succ x_i, w_i)$ be the current formula;
 4.    for $j$:= 1 to $i-1$ do
 5.       Initialize array $r'[1:i-1]$ to $0^{i-1}$;
 6.       for $k$:= 1 to $i$ do
 7.          if $k > j$ then set $r'[j]$+= $w_i * r[j]$;
 8.          else set $r'[j]$+= $1 * r[j]$;
 9.    $r$:= $r'$;
10. return $r[1]$.
```

Figure 2: Computing the partition function of a chain model.

Arbitrary Atomic Weighted Formulas

Suppose we have an arbitrary model on atomic weighted formulas, and we want to compute the partition function in order to compute some (conditional) probability of interest. First, we select a total ordering of the elements. The efficiency, but not the correctness, will depend on this order.

Define a *permutation generating tree* on ordered elements $x_1 \dots x_n$, recursively. For i from 1 to n, a tree of depth i is built as follows: the root of the tree consists of the element x_1, and the leaves consist of all permutations of the elements $x_1 \dots x_i$. To recursively extend the leaves by the next element, x_{i+1},

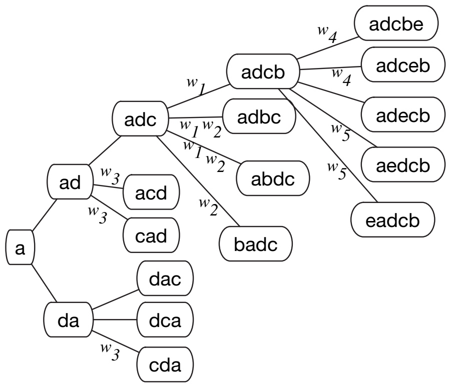

Figure 3: Part of a permutation generating tree for the formulas from Example 2. Elements are in order $[a,d,c,b,e]$; formulas are evaluated as soon as possible. Note that some arcs have two weights (w_1 and w_2), because two formulas ($a \succ b$ and $b \succ c$) become true when b is added to the order.

there is a child for each of the positions where the element could be inserted.

Arcs have weights induced by the weighted formulas. Whenever a weighted formula can be evaluated by a permutation, the weight is applied to the corresponding arc. A weighted formula is only evaluated once in any path. Figure 3 shows part of a permutation generating tree with arc weights corresponding to the weighted formulas in Example 2 (the ordering was chosen arbitrarily).

We could compute the partition function by starting at the leaves, summing over every branch, and multiplying by the weights as encountered. This would be an $O(n! * n)$ operation, so it is not practical. It can be, however, if we detect *symmetries*: cases where the operations would be identical and so only need to be done once. In the previous section, we outlined an algorithm that exploits the fact that it is only the count of the ancestors that have particular properties, not the identity of the elements that matters.

We will thus derive a variable elimination algorithm that eliminates the elements in the order $x_n \dots x_1$, and records as much information as necessary to compute the partition function. Note that we use the reversed ordering of elements than that assumed for the evaluation of the permutation-generating tree. The proposed algorithm will mimic the algorithm that evaluates a permutation-generating tree from the leaves, but groups together all the subtrees with identical counts. Towards this end, we generalize the concept of counting table presented in the previous section. Since, in the general case, the removal of a single element can affect more than one statement, the counting table must be more complex. We first need to define the following concepts:

Define P^n_i to be $n!/(n-i)!$, which is $n*(n-1)*\dots*(n-i+1)$. This is the number of ways i items can be selected in order from n items, which is like $\binom{n}{i}$ but where the *order* of the i items counts. This is equal to $\binom{n}{i}i!$; note that $P^n_i \leq n^i$.

Partial permutations. Let X and Y be sets of elements with $Y \subset X$; we say that any partial injective function $p : [1..m] \to Y$, where $m = |X|$, is a *partial permutation* of the elements in set X relative to Y. The concept of satisfaction of a preference formula by a partial permutation is defined as expected. The extension of a partial permutation p by element $x \in X \setminus Y$ in position i, denoted $p \oplus (x,i)$, creates a new partial permutation p' such that:

$$p'(j) = \begin{cases} p(j) & \text{if } 1 \leq j < i \\ x & \text{if } j = i \\ p(j-1) & \text{if } j > i\,. \end{cases}$$

```
Algorithm ComputeZ-Gen($M$)
Input: Weighted preference model $M$ over $n$ elements.
Output: The value of the partition function for $M$.
 1. Let $x_1,\dots,x_n$ be an arbitrary ordering of vars. in $M$;
 2. Initialize array $s[1:n]$ to $1^n$; let $V_{n+1} = \emptyset$;
 3. for $k := n$ downto 2 do
 4.     let $A_k$ be the set of formulas that contain $x_k$;
 5.     let $U_k$ be the set of elements $A_k$ (without $x_k$);
 6.     $M := M \setminus A_k$;
 7.     let $V_k = \{x_j \mid x_j \in U_k\} \cup (V_{k+1} \setminus \{x_k\})$;
 8.     let $¶_k$ be the set of partial perm. of $U_k$ w.r.t. to $V_k$;
 9.     Initialize array $s'[1:|¶_k|]$ to $1^{|¶_k|}$;
10.     for each $p_i \in ¶_k$ do
11.         for $j := 1$ to $k$ do
12.             for each $(f, w_f) \in A_k$ do
13.                 if $p_i \oplus (x_k, j) \models f$ then set $s'[i]$+= $w_f * s[e_{i,j}]$;
14.                 else set $s'[i]$+= $1 * s[e_{i,j}]$;
15.     $s := s'$;
16. return $s[1]$.
```

Figure 4: Algorithm for arbitrary atomic formulas.

The *counting table* is now a representation of a function from partial permutations into reals. Algorithm ComputeZ-Gen (Figure 4) shows how to construct counting table s' based on counting table s that was built during the previous iteration. The procedure starts by initializing a counting table s for element x_n; each item in the table corresponds to the position of element x_n relative to the other $n-1$ elements, that is, $s[1]$ corresponds to a partial permutation of the form $[x_n - - \dots -]$, where there are $n-1$ elements "$-$", $s[2]$ corresponds to $[-x_n - \dots -]$, and so on. We initialize each position with value 1. The sets of elements V_k accumulate the elements that have already been used in counting tables in previous iterations; clearly, V_{n+1} must be initialized to $\emptyset$.

As mentioned before, elements in M will be removed in order $x_n,\dots,x_1$; to remove element x_k, the algorithm computes the following sets: (i) A_k, the formulas in M that involve x_k, formally: $A_k = \{x_j \mid x_j \succ x_k \in M \; or \; x_k \succ x_j \in M\}$ (note that these formulas are taken out of consideration for later iterations), and (ii) U_k containing the elements that appear in A_k different from x_k. Finally, the set V_k is composed by the union of the elements in U_k and the elements that have been seen so far (V_{k+1}), not including x_k. In line 8, we compute all partial permutations of elements in U_k relative to V_k; this means that we consider all possible positions of elements in U_k, with respect to the rest of elements in

V_k, but leaving them unnamed. For each partial permutation $p_i \in \P_k$ (note that the size of $\P_k$ is $P^{k-1}_{|V_k|}$), the algorithm tries every possible extension by element x_k. For each of these, the product of the weights of all formulas in A_k that satisfy the extension is computed (if none are satisfied, this product is 1), and multiplied by the value $s[e_{i,j}]$, which corresponds to the entry in counting table s for the disposition of the elements given by $p_i \oplus (x_k, j)$. Finally, entry $s'[i]$ is updated by summing the terms obtained for each extension. The following example illustrates this process in detail.

Example 7. Consider the model from Example 2 with the order $[a,d,c,b,e]$; the elements are thus eliminated in order $[e,b,c,d,a]$. The algorithm proceeds as follows:

Eliminate element e: Counting table s is initialized with every partial permutation of $\{e\}$ relative to $\{e,b,c,d,a\}$, with value 1; that is, $s[1] = 1$ corresponds to $[e - - - -]$, ..., $s[5] = 1$ corresponds to $[- - - - e]$. The set A_5 contains formulas f_4 and f_5; thus, as s only contains element e, we have that $V_5 = \{c,d\}$. We then build the set of partial permutations $\P_5$ (of size $P^{k-1}_{|V_k|} = P^4_2 = 12$):

$s'[1]$	$cd - -$	$w_5 + w_4 * w_5 + 3 * w_4$
$s'[2]$	$c - d -$	$w_5 + 2 * w_4 * w_5 + 2 * w_4$
$s'[3]$	$c - - d$	$w_5 + 3 * w_4 * w_5 + w_4$
$s'[4]$	$-c\ d-$	$2 * w_5 + w_4 * w_5 + 2 * w_4$
$s'[5]$	$-c - d$	$2 * w_5 + 2 * w_4 * w_5 + w_4$
$s'[6]$	$- - cd$	$3 * w_5 + w_4 * w_5 + w_4$
$s'[7]$	$dc - -$	$w_5 + w_4 + 3$
$s'[8]$	$d - c -$	$w_5 + 2 * w_4 + 2$
$s'[9]$	$d - - c$	$w_5 + 3 * w_4 + 1$
$s'[10]$	$-d\ c-$	$2 * w_5 + 2 * w_4 + 1$
$s'[11]$	$-d - c$	$2 * w_5 + w_4 + 2$
$s'[12]$	$- - dc$	$3 * w_5 + w_4 + 1$

The third column above is obtained by assigning to each permutation in $\P_5$ the corresponding value using the equations in lines 13 and 14. For example, the number associated with $[c\ d - -]$ ($s'[1]$), is computed by evaluating the following extensions of $[c\ d - -]$ by e:

$$[e\,c\,d - -],\ [c\,e\,d - -],\ [c\,d\,e - -],\ [c\,d - e -],\ [c\,d - - e].$$

For each one, we add w_5, w_4, $w_5 * w_4$, or 1, depending on whether the permutation satisfies f_5 but not f_4, f_4 but not f_5, both f_4 and f_5, or neither formula, respectively. We get $s'[1] = w_5 * s[1] + w_4 w_5 * s[2] + w_4 * (s[3] + s[4] + s[5])$, since $[ecd - -]$ satisfies f_5 only, $[ced - -]$ both, and the other three satisfy only f_4. Note, here, the $s[e_{i,j}]$ terms are always 1.

Eliminate element b: we have two formulas that contain element b, namely f_1 and f_2. The set of elements affected by deleting b is $\{a,c\}$. We need to build the following set of partial permutations (of size $P^3_2 = 6$): $[ac-]$, $[a-c]$, $[-ac]$, $[ca-]$, $[c-a]$, $[-ca]$. The weights for the partial permutations are obtained as follows. To compute the value corresponding to partial permutation $[ac-]$, we consider the extended permutations by b:

$$[b\,a\,c\,-],\ [a\,b\,c\,-],\ [a\,c\,b\,-],\ [a\,c\,-\,b].$$

In this case, the term associated with extended permutation $[bac-]$ not only depends on whether it satisfies formulas f_1 and f_2 (i.e, the relationship among the positions of b, c, and a), but it also depends on the satisfaction of the formulas that we already processed (i.e., the position of the rest of the elements); this is provided by table s. We have that $[bac-]$ corresponds to permutation $[- - cd]$ from the previous iteration; note that element d can only occupy the last position. Therefore, for partial permutation $[bac-]$, we add the term $w_2 * s([- - cd]) = w_2 * (3 * w_5 + w_4 w_5 + w_4)$ (factor w_2 is due to the fact that it satisfies only formula f_2), for $[abc-]$ the term $w_1 w_2 * s([- - cd])$, for $[acb-]$ the term $w_1 * s([-c-d])$, and finally, the term $w_1 * s([-cd-])$ corresponding to $[ac-b]$. We then have: $s'([ac-]) =$

$$w_2 * s([- - cd]) + w_1 w_2 * s([- - cd]) + \\ w_1 * s([-c-d]) + w_1 * s([-cd-]) =$$

$$w_2 * (3w_5 + w_4 w_5 + w_4) + w_1 * w_2 * (3w_5 + w_4 w_5 + w_4) + \\ w_1 * (2w_5 + 2w_4 w_5 + w_4) + w_1 * (3w_5 + w_4 w_5 + w_4).$$

The values for the rest of the permutations are computed similarly, and s is replaced by s'.

Eliminate element c: this is the last step, as there is only one formula left (f_3), and it contains element c. The only element affected by eliminating c at this point is d. Thus, we have only two partial permutations to consider: $[d\,-]$, $[-\,d]$.

Now, $s'([d-]) = w_3 * s([c-a]) + s([-ca]) + s([-ac])$ and $s'([-d]) = w_3 * s([ca-]) + w_3 * s([ac-]) + s([a-c])$.

The value $s([d-]) + s([-d])$ then yields Z. ■

The complexity of this algorithm can be described in terms of the structure and the elimination order chosen. Given a model and an element ordering $x_1, \ldots, x_n$, define the *linear cut size for element* x_k, denoted s_k, as the number of elements x_j, with $j < k$, that are connected to an x_l where $k \leq l$; that is,

$$s_k = \big|\{x_j \mid j < k \text{ and } \exists l \text{ such that } k \leq l \text{ and } \\ x_j \succ x_l \text{ or } x_l \succ x_j \text{ appear in } M\}\big|.$$

These x_j are the elements that make up the counting table when x_k is eliminated. We then define the *linear cut size* (denoted c) of a model and ordering as the maximum, over positions k in the ordering, of s_k.

Proposition 8. *The complexity of Algorithm* ComputeZ-Gen *is $O(m * n^2 * n^c)$, where m is the number of statements, n is the number of elements, and c is the linear cut size.*

Proof. The `for` loop in line 3 is executed $n - 1$ times; within that loop, line 4 requires $O(m)$ time.

Now, suppose that an arbitrary element x_k (with $2 \leq k \leq n$) is being eliminated. The number of times the loop in line 10 is repeated (and the cost of the initialization in line 9) depends on the number of partial permutations in $\P_k$, which is $P^{k-1}_{|V_k|}$, where V_k contains all elements x_j (with $j < k$) that are connected to an x_l where $k \leq l$. That is, $|V_k|$ is s_k, which is the cut size for x_k. Therefore, $|\P_k| = P^{k-1}_{|V_k|} = P^{k-1}_{s_k} \leq (k-1)^{s_k}$. This is the size of the counting table that is built for element x_k. Clearly, since $k \leq n$, and c is by definition the maximum of all s_k's, then n^c is an upper bound for the size of the largest counting table that has to be built.

For each partial permutation in $\P_k$, k positions must be tested in order to compute its corresponding value—the `for` loop in line 11 is executed $O(n)$ times, and thus the loop in line 10 requires $O(n*n^c)$ time in total. Note that the instruction inside `for` loop in line 12 are executed m times *in total*, and thus the cost of this loop is absorbed by the $O(m)$ term. Putting it all together, we get a total of $O(m*n^2*n^c)$. □

Proposition 8 tells us that our algorithm is fixed-parameter tractable—if we can bound c, the asymptotic cost behaves similar to the chain case (for chains, we have $m = n-1$).

A Heuristic Elimination Order. As we mentioned before, the total order selected among the elements is not important from the point of view of the correctness of our algorithm, but it nevertheless has a great impact on the efficiency of the process (cf. the experimental results). The critical part in the algorithm is the construction of the counting table; as described above, for a model with n elements and assuming we are eliminating element x_k (thus, $k-1$ remain), the size of this counting table is $P^{k-1}_{|V_k|}$. Clearly, a careless choice in the elimination order could have a great impact on the size of the tables that need to be built. A simple heuristic to try and minimize this effect is to select at each step the element that is involved in the least possible number of formulas (among the remaining ones). Essentially, this generalizes the approach for chains; though, in general, the structure is more complex, it looks for a piece of the model that is as close as possible to a chain, so the algorithm needs to consider the relative positions of as few elements as possible.

Example 9. Return to the same setup as in Example 2, but suppose that we now apply the heuristic to choose in which order the elements are eliminated; in this case, we construct quite different data structures. At the beginning, we have the following counts of how many formulas each element appears in: $\#a = 1, \#b = 2, \#c = 3, \#d = 2, \#e = 2$.

The heuristic thus chooses to eliminate a first; then, we only have to consider every position that element b can have in a permutation of size 4, of which there are 4. Further, as in all cases of elements with a count of 1, we can simply store the count when b is in each position as we do for chains. We then eliminate element b and, continuing, we can see that we never need to build a table larger than $P^3_2 = 6$.

In Example 7, we started with element e; if we had started with c, the table would have been of size $P^4_3 = 24$. ■

Experimental Evaluation

We developed a prototype implementation of the weighted orderings framework and associated algorithms consisting of about 3,200 lines of Java code. We now report on the results of a series of experiments designed to test how well our variable elimination algorithms scale. All experiments were performed on a computer running the Windows 7 SP1 64-bit Professional Edition OS with a dual-core Intel Core i5-4210M processor at 2.6/3.2 GHz and 16GB of physical RAM (SODIMM DDR3 at 1,600MHz); 12GB of RAM were made available to the Java virtual machine's heap, and the system load was minimized in order to guarantee that the entire heap fit in physical memory—therefore, no swapping to disk was necessary. Multiple runs were performed in each case to minimize experimental error.

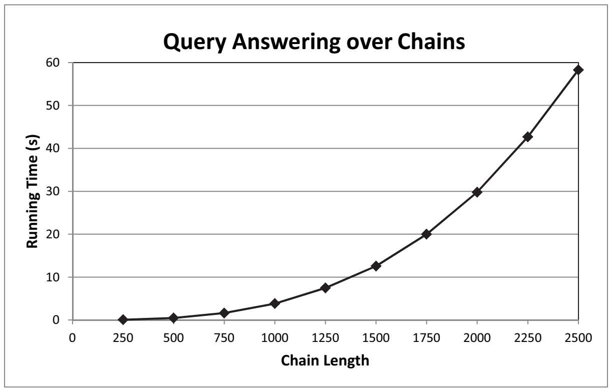

Figure 5: Running time evaluation of the chain algorithm.

The Chain Case. The first experiment focuses on the case in which the input model is a chain, as described by Algorithm 2. For these runs, all queries were randomly generated and of size 2 (i.e., two atomic conjuncts), and the numbers reported are averages over 10 runs. Figure 5 shows how the variable elimination algorithm for chains scales to models with 2,500 elements in under a minute; the theoretical polynomial running time is clearly reflected in the shape of the curve. This is much better than the algorithm of (Lukasiewicz, Martinez, and Simari 2014), which is exponential in the number of preference statements.

The General Case. These runs were performed over inputs with 20 elements and queries of size 1. Figure 6 shows four plots, varying the number of statements and measuring running time and maximum counting table size. For the non-heuristic case, we performed up to 10 runs for models of 1-25 statements (in steps of 1); for the heuristic, the same was done for models of up to 50 statements (in steps of 5 from 25 to 50). These results illustrate the huge impact of the heuristic, allowing to double the size of the model and still answer queries much faster than the non-heuristic algorithm.

The great variability in these plots is due to the fact that our algorithm's cost is given by the maximum size of the linear cuts encountered; thus, different models of the same size can vary greatly in the complexity of their structure (compare a chain with a clique). Nevertheless, the number of runs performed allows to establish a trend as the size of the models grows. Another interesting observation is that this variability is reduced in the heuristic version—this is because the algorithm chooses the easiest cases first, and thus costly cases become rarer than in the non-heuristic runs.

Related Work

As was mentioned in the introduction, this work continues our initial proposal in (Lukasiewicz, Martinez, and Simari 2014), where both exact and approximate algorithms were introduced for general (that is, not necessarily atomic) weighted preference statements. The algorithms in that paper are based on leveraging (exact and approximate) algorithms that count linear extensions of linear orders; in order for this to be possible, the resulting algorithms partition the

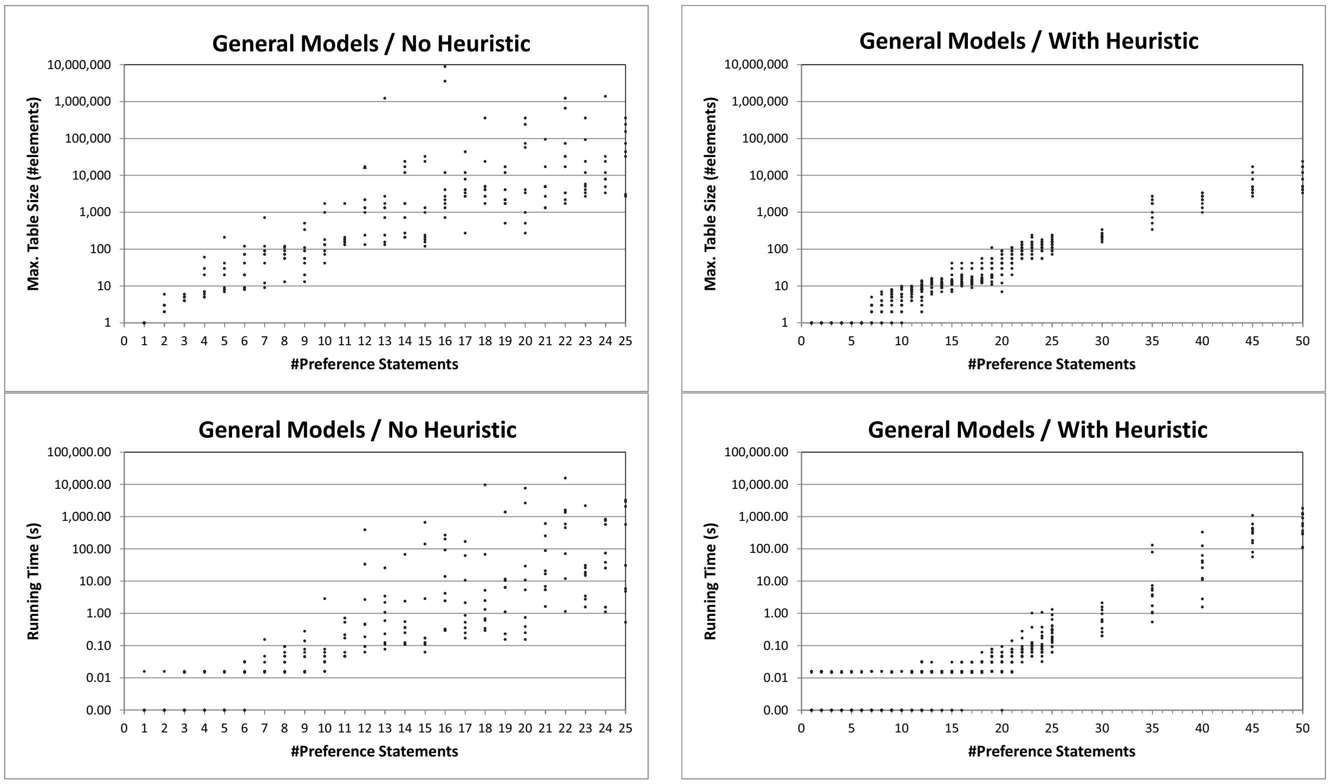

Figure 6: Results for randomly generated atomic models; all Y-axes are on a logarithmic scale. *Left:* results for the non-heuristic algorithm; *Right:* results for the heuristic version. Note that the X-axes for the heuristic go up to 50, while the ones for the non-heuristic only reach 25. *Top:* maximum sizes that the counting tables reached; *Bottom:* running times.

universe of possible worlds into equivalence classes, which is where the exponential term in the number of statements arises. Those algorithms are thus inherently different from the ones explored here, since they are geared towards models with bounded number of general statements (that can still have large numbers of elements, if the statements are complex). Thus, as stated before, those algorithms have no chance of scaling well when there are many statements in the models, like the ones in our experimental evaluation.

One of the first modern works to propose a model of probabilistic distributions of linear orderings is quite early and comes from the statistics literature (Mallows 1957); it assumes that there is a "true" ranking, and the rest of the orderings have a probability that is proportional to their distance to this ranking. Interestingly, in the 18th century Condorcet studied—from the perspective of social choice—the related problem of finding a ranking that is most likely to be correct (again, relative to a reference ranking); essentially, he showed that this is the ranking that minimizes the number of disagreements with the true ranking (de Condorcet 1785). Another well-known model was proposed some years later in (Plackett 1975) and is now commonly known as the Plackett-Luce model. Though it is more general than the Mallows model, this formalism is based on weighting individual elements, and thus cannot represent the more general weighted statements (expressing relations between elements) used here. These works are thus geared towards obtaining statistical tools (not focused on computational efficiency) to perform estimations based on noisy models, rather than the more general and tractable inference tools that we pursue here. They have been applied in recent works in machine learning to build simpler models in which ties are permitted (i.e., rankings are not total) (Lu and Boutilier 2011), or in non-parametric versions where sampling can be done more efficiently, but inference procedures are not explored (Lebanon and Mao 2008). Other attempts to represent distributions over linear orderings in a compact fashion such as (Huang, Guestrin, and Guibas 2009) also fail in providing tractable exact inference.

The recent work of (Choi, Van den Broeck, and Darwiche 2015) is quite close in spirit to our approach; they leverage recent developments of tools for learning structured probability spaces (Kisa et al. 2014) towards obtaining a more general framework for learning probabilistic preferences. Their model allows complex queries and tractable learning; however, even though the authors experimentally show that partial rankings (with ties) can be handled efficiently, they also show that their approach does not scale well with total rankings of the kind adopted in this paper.

Summary and Outlook

In this work, we have developed exact algorithms for probabilistic query answering over weighted orderings; though our ideas are inspired in well-known variable elimination approaches, adapting these techniques to these models required developing completely novel algorithms. The first algorithm presented is designed for chain models, while the second—a generalization of the first—works for any set of atomic preference statements in which the query and evidence are conjunctions of atomic preference formulas. The former is guaranteed a cubic running time, while the latter is polynomial with a degree depending on the maximum cut size of the model and chosen variable elimination order; the algorithm is thus tractable, if the cut size can be bounded.

Finally, we presented empirical results showing that the algorithms scale to reasonably-sized models; our results also show that further optimizations would go a long way in affording practical query answering over larger inputs. In particular, finding a way to reduce the size of the counting tables would greatly help—we are currently investigating if the combinatorial term can be reduced by identifying cases in which counting table entries do not need to be combined and can thus be computed separately more efficiently.

Acknowledgments

This work was supported by the UK EPSRC grants EP/J008-346/1, EP/L012138/1, and EP/M025268/1, and funds from Universidad Nacional del Sur and CONICET, Argentina.

References

Bertelè, U., and Brioschi, F. 1972. *Nonserial dynamic programming*, volume 91 of *Mathematics in Science and Engineering*. Academic Press.

Bozga, M., and Maler, O. 1999. On the representation of probabilities over structured domains. In *Proc. of CAV*, 261–273.

Choi, A.; Van den Broeck, G.; and Darwiche, A. 2015. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proc. of IJCAI*, 2861–2868.

Chomicki, J. 2003. Preference formulas in relational queries. *ACM Trans. Database Syst.* 28(4):427–466.

Darwiche, A. 2001. Recursive conditioning. *Artif. Intell.* 126(12):5–41.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

de Condorcet, N. 1785. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'Imprimerie Royale.

Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. of UAI*, 211–219.

Huang, J.; Guestrin, C.; and Guibas, L. 2009. Fourier theoretic probabilistic inference over permutations. *J. Mach. Learn. Res.* 10:997–1070.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *Proc. of KR*, 558–567.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Lauritzen, S. L., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B* 50(2):157–224.

Lebanon, G., and Mao, Y. 2008. Non-parametric modeling of partially ranked data. *J. Mach. Learn. Res.* 9:2401–2429.

Lu, T., and Boutilier, C. 2011. Learning Mallows models with pairwise preferences. In *Proc. of ICML*, 145–152.

Lukasiewicz, T.; Martinez, M. V.; and Simari, G. I. 2014. Probabilistic preference logic networks. In *Proc. of ECAI*, 561–566.

Mallows, C. L. 1957. Non-null ranking models. *Biometrika* 44(1/2):114–130.

Plackett, R. L. 1975. The analysis of permutations. *Applied Statistics* 24(2):193–202.

Raedt, L. D.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic Prolog and its application in link discovery. In *Proc. of IJCAI*, 2462–2467.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Mach. Learn.* 62(1/2):107–136.

Rossi, F.; Venable, K. B.; and Walsh, T. 2011. *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan & Claypool Publishers.

Roth, D. 1996. On the hardness of approximate reasoning. *Artif. Intell.* 82(1/2):273–302.

Zhang, N. L., and Poole, D. 1994. A simple approach to Bayesian network computations. In *Proc. of Canadian AI*, 171–178.