Elimination Ordering in Lifted First-Order Probabilistic Inference

Seyed Mehran Kazemi and **David Poole**
The University of British Columbia
Vancouver, BC, V6T 1Z4
{smkazemi, poole}@cs.ubc.ca

Abstract

Various representations and inference methods have been proposed for lifted probabilistic inference in relational models. Many of these methods choose an order to eliminate (or branch on) the parameterized random variables. Similar to such methods for non-relational probabilistic inference, the order of elimination has a significant role in the performance of the algorithms. Since finding the best order is NP-complete even for non-relational models, heuristics have been proposed to find good orderings in the non-relational models. In this paper, we show that these heuristics are inefficient for relational models, because they fail to consider the population sizes associated with logical variables. We extend existing heuristics for non-relational models and propose new heuristics for relational models. We evaluate the existing and new heuristics on a range of generated relational graphs.

Introduction

Probabilistic graphical models, including Bayesian networks and Markov networks (Pearl 1988) are representations for dependencies among random variables.

Even though exact inference in these models is known to be NP-hard (Cooper 1990), variable elimination (Zhang and Poole 1994) and recursive conditioning (Darwiche 2001) are two simple algorithms for exact inference which are efficient for graphs with a low treewidth. The former is a dynamic programming approach and the latter is a search-based method. Both of these algorithms select an order for the variables and eliminate (branch on) the variables in that order to find the result of the given query. The elimination order for variable elimination is the reverse of the splitting (branching) order for recursive conditioning with the same additions and multiplications (Darwiche 2001); here we will refer to the elimination ordering even if we are doing search.

The performance of these methods is highly dependent on the elimination order. Since finding an order which results in the minimum total state space or in a width equal to the treewidth of the graph is NP-complete (Arnborg, Corneil, and Proskurowski 1987; Koller and Friedman 2009), in practice heuristics are used to choose the order. Among these heuristics, minimum-fill, minimum-size and minimum-weight are known to produce good orderings (Sato and Tinney 1963; Kjaerulff 1985; Dechter 2003). These heuristics aim to minimize the number of introduced fill-edges, the size of the largest clique and the weight of the largest clique respectively. Other methods in the literature to choose an elimination ordering include maximum cardinality search (Tarjan and Mihalis 1984), LEX M (Rose, Tarjan, and Lueker 1976) and MCS-M (Berry et al. 2004). There are also heuristics with local search methods such as simulated annealing (Kjaerulff 1985), genetic algorithm (Larranaga et al. 1997) and tabu search (Clautiaux, Ngre, and Carlier 2004).

Relational probabilistic models (Getoor and Taskar 2007) or template-based models (Koller and Friedman 2009) allow for probabilistic dependencies among relations of individuals. They extend Bayesian networks and Markov networks by adding the concepts of objects, object properties and relations. One of the major challenges in these models is to do lifted inference (inference without grounding out the representation). Inference on the lifted level can potentially dramatically reduce the number of computations. Exact inference is also the basis for many approximate inference algorithms, either as a subcomponent (e.g., in sampling methods) or as an ideal to approximate (e.g., in variational methods).

Poole (2003) proposed the problem of lifted inference in these models. He introduced parfactor representation, the concept of splitting, and proposed a lifted algorithm for multiplying factors and summing out variables. de Salvo Braz, Amir, and Roth (2005) realized that in order to do lifted inference, the identity of individuals is not important, but it is the number of individuals having a property which is important. Based on this, they introduced counting elimination by which lifted inference could in many cases be done in polynomial time and space rather than exponential in the number of individuals in grounding. Milch et al. (2008) proposed the C-FOVE algorithm which allowed for more cases when counting was applicable by introducing counting formulas for representing the intermediate results of counting.

Gogate and Domingos(2010), Jha et al. (2010), den Broeck et al. (2011) and Poole, Bacchus, and Kisynski (2011) proposed search-based lifted inference, which also allows us to exploit logical structure.

All of these algorithms are very sensitive to the elimination (splitting) order. Some of them have constraints on elimination orderings for correctness, but still leave many choices of possible orderings. None of the papers compared multiple elimination orderings. Just like non-relational models, the elimination order can severely affect the performance of the algorithm for relational models.

In this paper, we propose and evaluate some heuristics for elimination ordering in relational models. First, we bring some examples which show that the elimination ordering heuristics used for non-relational graphical models are not suitable for relational models. Then we examine the problems with these heuristics and propose other heuristics to solve these problems. Since there have been no attempts to optimize the elimination ordering for relational models, we compare our results with the best heuristics for non-relational models.

Background

Relational probabilistic models represent probability distributions over relations among individuals.

A **population** corresponds to a domain in logic and refers to a set of **individuals**. The cardinality of the population is called **population size** which is a non-negative number. For example, a population can be the set of soccer teams in *world cup 2014*, where *Germany* is an individual and the population size is *32*.

Logical variables start with lower-case letters, and **constants** start with an upper-case letter. Each logical variable x has a type τ and a population $pop(x)$ associated with it where $|x| = |\tau| = |pop(x)|$ is the size of the population.

A **parameterized random variable (PRV)** (Poole 2003) is of the form $F(t_1,...,t_k)$ where F is a k-ary predicate symbol and each t_i is a logical variable or a constant. If $k = 0$ we can omit the parentheses. For example, $PlaysFor(player, team)$ can be a PRV with predicate symbol $PlaysFor$ which is true if $player$ plays for $team$. We use capital letters in bold to represent a set of PRVs.

A **parametric factor** or **parfactor** (Poole 2003) is of the form $\langle C, \mathbf{V}, \phi \rangle$ where $\mathbf{V}$ is a set of PRVs, C is a set of constraints on the logical variables in $\mathbf{V}$ and ϕ is a factor (which is a function from assignments of values to PRVs in $\mathbf{V}$, into non-negative real numbers). As an example, $\langle \{x \neq y\}, \{Friend(x,y), Adult(x), Adult(y)\}, \phi \rangle$ is a parfactor having three PRVs with two logical variables that are not equal, and a function ϕ which maps every set of assigned values to the three PRVs (e.g. $Friend(x,y) = True, Adult(x) = True, Adult(y) = False$) into a non-negative real number. A probability model can be defined by a set of parfactors.

A **parfactor graph** consists of a set of parfactors as the nodes of the graph where there is an arc between two parfactors f_1 and f_2 if there is a PRV V_1 in f_1 which unifies with a PRV V_2 in f_2 without violating any of the constraints of the two parfactors.

A **grounding** of a relational model is a model consisting of all instances of the PRVs where logical variables are replaced by individuals in their population such that the constraints are satisfied.

$\mathbf{V_1(x)}$	**count**
True	i
False	$n-i$

(a)

$\mathbf{V_1(x)}$	$\mathbf{V_2(x)}$	**count**
True	*True*	j
True	*False*	$i-j$
False	*True*	k
False	*False*	$n-i-k$

(b)

Figure 2: Two counting contexts

Search-Based Lifted Inference

Gogate and Domingos (2010), Jha et al. (2010), den Broeck et al. (2011) and Poole, Bacchus, and Kisynski (2011) proposed search-based lifted inference algorithms. We base our work on lifted recursive conditioning (LRC) (Poole, Bacchus, and Kisynski 2011) because the algorithm was straightforward to implement, and the differences from the others were small enough so that what is learned about elimination ordering is applicable to all of the algorithms. We first give the definitions for *counting context* and *context* which are used in the inference algorithm and then describe the algorithm for Boolean PRVs.

A **counting context** is of the form $\langle \mathbf{V}, \chi \rangle$, where $\mathbf{V}$ is a set of PRVs, and χ is a function mapping assignments of PRVs in $\mathbf{V}$ into non-negative integers. It specifies how many tuples of individuals in the cross product of the population of the logical variables take on the values for each assignment in $\mathbf{V}$. Figures 2(a) and 2(b) represent two counting contexts. The former indicates $V_1(x) = True$ for i out of n individuals of x and is $False$ for the rest, and the latter indicates $V_1(x) \wedge V_2(x) = True$ for j out of n individuals of x, $V_1(x) \wedge \neg V_2(x) = True$ for $i-j$ out of n individuals of x, and so forth.

A **context** is a set of counting contexts such that the logical variables in the PRVs in separate counting contexts represent different populations. A PRV is **assigned** in a context if it unifies with a PRV in one of the counting contexts. For example, $V_1(x)$ is assigned in a context having any of the counting contexts in Figure 2. A parfactor is **assigned** in a context if all its PRVs are assigned in that context.

The LRC algorithm is a recursive function which takes in a context *con* and a set of parfactors Fs and calculates the product of the values of the parfactors in that context by recursive calls. Computed values are stored in a cache to avoid recomputation for the same recursive call. The function is first called by an empty context and the set of all parfactors in the model. The recursion stops if there are no parfactors (i.e. $Fs = \{\}$), and 1 is returned. If Fs with *con* has already been computed and is in the cache, that value is returned. If any parfactor $f \in Fs$ is assigned in *con*, it is evaluated and its value is multiplied by the value of the recursive call with the other parfactors (i.e. $Fs \backslash f$). If the parfactor graph is disconnected, a recursive call is made for each connected component and the product of the returned values is returned. If a logical variable x is in all PRVs that are not assigned in *con* in all parfactors (perhaps renamed), the grounding is a set of identical (up to renaming) disconnected components, one for each assignment of x; we recursively evaluate one component and raise it to the power of the $|pop(x)|$. Otherwise a

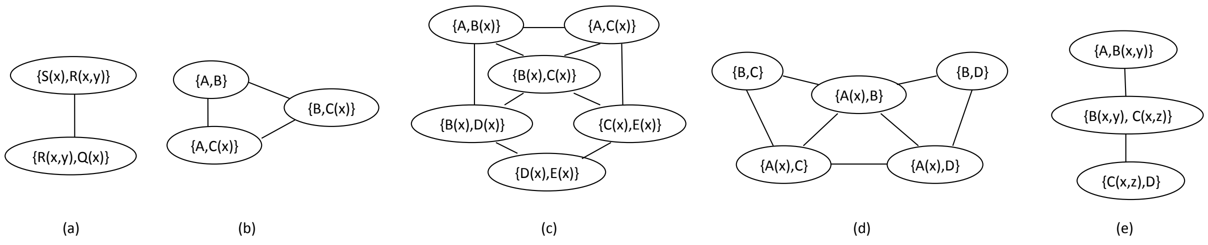

Figure 1: Parfactor graphs: (a) Example 1, (b) Example 2, (c) Example 3, (d) Example 4, (e) Example 5.

PRV V is selected to branch on. Branching on a PRV is performed by making recursive calls to the LRC function; the number of generated branches corresponds to the number of recursive calls.

A PRV V having no logical variables generates two branches (i.e. two recursive calls). In one branch, a counting context assigning V to $False$, and in the other one, a counting context assigning V to $True$ is added to the context con.

A PRV V_1 having a logical variable x of type τ (where we still have not branched on any other PRV having a logical variable of type τ) generates $n+1$ branches where $n = |pop(x)|$. In the i-th branch, a counting context assigning $V_1 = True$ to i out of the n individuals of x and the other $n-i$ individuals to $False$ (as in Figure 2 (a)) is added to the context. The value of this branch is multiplied by $C(n,i)$ because that's the number of ways we can have i *True* individuals out of a population of n.

Suppose we have already branched on a PRV V_1 having a logical variable of type τ and we want to branch on a PRV V_2 having a logical variable of the same type. Consider the case for V_1 with i *True* individuals (as in Figure 2 (a)). In order to branch on V_2, we first generate $i+1$ branches where j-th branch represents the case when for the i *True* individuals in V_1, j of them have V_2 *True* and the other $i-j$ have V_2 *False*, and the value of this branch is multiplied by $C(i,j)$. For each branch, we generate $n-i+1$ branches where k-th branch represents the case when for the $n-i$ *False* individuals in V_1, k of them have V_2 *True* and the other $n-i-k$ have V_2 *False*, and the value of this branch is multiplied by $C(n-i,k)$. The counting context for V_1 in con (represented in Figure 2(a)) is then replaced by the counting context in Figure 2 (b).

The case where we want to branch on a PRV V of type τ where we have already branched on m PRVs having a logical variable of type τ is similar to the previous case where for each assignment of values to the previous m PRVs, we generate a level of branches assigning the number of *True* and *False* individuals of V.

If the PRV selected to be branched on has more than one logical variable, one of them is grounded and the algorithm works as described.

Example 1. Consider a parfactor graph with two parfactors where $\mathbf{V_1} = \{S(x), R(x,y)\}$ and $\mathbf{V_2} = \{R(x,y), Q(x)\}$ and suppose x and y have different types and there is no constraint on the parfactors (Figure 1 (a)). Suppose that the order in which we want to branch on PRVs is $\langle S, R, Q \rangle$. First, the algorithm realizes that the grounding is disconnected because all PRVs of all parfactors have the same logical variable x. So it replaces x by one of its individuals X_i getting a new set of parfactors with $\mathbf{V_1} = \{S(X_i), R(X_i,y)\}$ and $\mathbf{V_2} = \{R(X_i,y), Q(X_i)\}$), does the calculations for this new set and raises the result to the power of $|pop(x)|$. In order to solve the problem with the new set of parfactors, it branches on S which makes two branches. Then for the first branch, it branches on R which produces $|1 + pop(y)|$ new branches. At this point, the algorithm evaluates the first parfactor and then branches on Q which produces two new branches. Then the second parfactor is evaluated and the results are stored in the cache. This process continues for other branches until the final result is obtained.

Existing Heuristics for Elimination Ordering

Most heuristics for elimination ordering in non-relational models try to minimize the width (number of random variables in a factor, which would correspond to the number of PRVs in a parfactor) of the largest generated factor. While this is appropriate for these models, it is not a good measure of complexity for lifted inference in relational models. Among existing heuristics for non-relational models, the minimum-fill greedy heuristic is the one which is widely used in practice (Dechter 2003). This heuristic successively eliminates a variable which produces the fewest number of fill-edges and often breaks the ties arbitrarily. Here we assume that ties are broken alphabetically. In this section, we bring some examples which show why this is not a good heuristic for relational models. Then we use the intuition from the examples to motivate new heuristics which address these problems.

In the rest of the paper, we refer to the order in which PRVs are eliminated which is the reverse order in which we branch on in search-based lifted inference. That's because the heuristic that we found that works best, as well as most existing heuristics in the literature and also other proposed heuristics in this paper (except one of them called "population order" heuristic) greedily select a PRV to eliminate. For simplicity, we assume the parfactors have no constraints and write them just by their set of PRVs. We use the simple term "min-fill" to refer to minimum-fill greedy heuristic.

Inefficiency of Existing Heuristics

Example 2. Consider a parfactor graph with parfactors $\{A, B\}$, $\{B, C(x)\}$ and $\{A, C(x)\}$ (Figure 1 (b)) and suppose

$|pop(x)| = n$. If we use min-fill to choose the elimination ordering, all PRVs produce zero fill-edges. By breaking the tie alphabetically, A is the first PRV in the order. However, eliminating A produces the parfactor $\{B, C(x)\}$ which needs $2 * (n+1)$ branches to be evaluated (2 branches for $B = True$ and $B = False$ and in each of these branches, $(n+1)$ branches for C where i-th branch represents the case where i of the individuals are $True$ and the rest are $False$). But if C was the first PRV in our order, eliminating C would result in the parfactor $\{A, B\}$ which needs only $2*2$ branches to be evaluated.

This example shows that the idea of breaking ties arbitrarily in min-fill is not a good choice for relational models and a better alternative is to break ties according to the population sizes of logical variables.

Example 3. Consider a parfactor graph with parfactors $\{A, B(x)\}$, $\{A, C(x)\}$, $\{B(x), C(x)\}$, $\{B(x), D(x)\}$, $\{C(x), E(x)\}$ and $\{D(x), E(x)\}$ (Figure 1 (c)) and suppose $|pop(x)| = n$. If we use min-fill, A is the first PRV eliminated because it's the only PRV that introduces no fill-edges. This means that in search-based lifted inference, A is the last PRV we branch on. Therefore, we will have a very huge tree when we branch on B, C, D and E (it will be a tree with a depth of 15).

Consider what happens when A is at the end of the elimination order. In this case, we first branch on A. Then, all PRVs have the same logical variable. This means we can solve the problem for one individual and raise the result to the power of n. In this way, we effectively have a non-relational problem.

This example represents the case with PRVs having logical variables with large population sizes. By placing these PRVs at the beginning of the elimination order (equivalently at the end of branching order), we may solve the problem just for one of the individuals which offers a significant computational benefit.

Example 4. Consider a parfactor graph with parfactors $\{A(x), B\}$, $\{A(x), C\}$, $\{A(x), D\}$, $\{B, C\}$ and $\{B, D\}$ (Figure 1 (d)) and suppose $|pop(x)| = n$. Using any method which tries to minimize the width of the largest generated parfactor, A is not the first PRV in elimination ordering. In min-fill, for example, C or D is the first PRV in the order depending on how we break the tie. However, eliminating C or D results in the parfactor $\{A(x), B\}$ which needs $(n+1)*2$ branches to be evaluated. Now suppose A is the first PRV in the order. In this case, eliminating A results in a parfactor $\{B, C, D\}$ which only needs $2*2*2$ branches.

This example shows that the width of the largest generated parfactor is not the only important factor which should be taken into account but the population sizes of the logical variables in its PRVs are also important. A parfactor with two PRVs having highly populated logical variables needs many more branches to be evaluated than a parfactor with three PRVs having no logical variables and avoiding to generate such parfactor should be more encouraged.

Example 5. Consider a parfactor graph with parfactors $\{A, B(x,y)\}$, $\{B(x,y), C(x,z)\}$ and $\{C(x,z), D\}$ (Figure 1 (e)) and suppose $|pop(x)| = |pop(y)| = |pop(z)| = n$. Using min-fill, the elimination ordering is $\langle A, B, C, D\rangle$ meaning that after branching on D, it branches on C. Since C has more than one logical variable, the algorithm grounds one of them. However, the elimination ordering $\langle B, C, A, D\rangle$ only branches on PRVs with fewer than two logical variables and does not require grounding any population.

This example demonstrates that it's a good idea to place the PRVs with more than one variable at the beginning of the elimination ordering (equivalently at the end of branching ordering) to avoid grounding the graph as much as possible.

Proposed Heuristics for Elimination Ordering

The examples in the previous section motivate the following intuitions: We should try to eliminate PRVs having variables with large population sizes sooner than others; we should pay attention to the size of the new parfactor generated rather than just its width; PRVs with more than one logical variable should be at the end of the branching ordering. Based on these observations, we propose new heuristics.

We define the *context-free branching factor (CFBF)* of a PRV $V(x_1, \ldots, x_m)$ to be $C(\prod_{i=1}^{m}(|pop(x_i)|) + |range(V)| - 1, |range(V)| - 1)$, where $C(i,j) = \frac{i!}{j!(i-j)!}$ and$|range(V)|$ is the number of values that V can take. Note that this formula corresponds to the number of branches when branching on the PRV V, assuming an empty context. Also note that as explained in search-based lifted inference section, the number of branches for a PRV depends on the context. The reason why the number of branches for the PRV V is equal to the given combination is that each branch represents a particular assignment of values in $range(V)$ to the individuals, so the number of branches is equal to the number of ways we can assign non-negative integer values to $|range(V)|$ numbers $K_1, K_2, \ldots, K_{|range(V)|}$ so that they sum to $\prod_{i=1}^{m}(|pop(x_i)|)$, and this can be solved by the given combination.

We also define the *context-free branching factor (CFBF)* of a parfactor to be $\prod_{i=1}^{n}(1 + |pop(PRV_i)|)$ where n is the number of PRVs in the parfactor and PRV_i is its i-th PRV. These two definitions are used in our proposed heuristics.

MinTableSize Heuristic

The *MinTableSize* heuristic places PRVs with more than one logical variable at the beginning of the elimination order sorted in descending order by the number of logical variables they contain, and among those with the same number, one with the largest CFBF is eliminated first. Then among PRVs having no more than one logical variable, it eliminates the PRV that results in a parfactor with minimum CFBF sooner. The idea of eliminating a PRV that results in a parfacotr with smaller CFBF sooner resembles the intuition behind the operation selection of (Milch et al. 2008), in which a cost is assigned to each possible operation by computing the total size of the factors generated by the operation and the one having the minimum cost is performed next. However, we only do this for PRVs with less than two logical variables and place the PRVs with at least two logical variables at the beginning of the elimination order to avoid grounding, and the only operation we have here is elimination (or branching). Using *MinTableSize* heuristic, ties may occur.

We break ties according to the CFBFs of the PRVs, where a PRV with a larger CFBF will be eliminated sooner.

This heuristic is consistent with our observations in the examples. The algorithm for selecting the next PRV to be eliminated using *MinTableSize* is shown in Algorithm 1. The first loop finds PRVs with more than one logical variable and selects one with maximum number of logical variables, then the maximum CFBF. If there is no such PRV, the second loop finds the PRV which produces the smallest parfactor and breaks the ties according to the CFBF.

Algorithm 1 *MinTableSize* algorithm

```
nextPRV $\longleftarrow \emptyset$
maxCFBF $\longleftarrow 0$
maxNumLogVars $\longleftarrow 2$
for $i = 1$ to numberOfPRVs do
    if numLogVars(PRV[i]) $>$ maxNumLogVars or (numLogVars(PRV[i]) $=$ maxNumLogVars and $|CFBF$(PRV[i])$| >$ maxCFBF) then
        maxNumLogVars $\longleftarrow$ numLogVars(PRV[i])
        maxCFBF $\longleftarrow |CFBF$(PRV[i])$|$
        nextPRV $\longleftarrow$ PRV[i]
if nextPRV $\neq \emptyset$ then
    return nextPRV
minTableSize $\longleftarrow +\infty$
for $i = 1$ to numberOfPRVs do
    f $\leftarrow$ parfactor resulting from eliminating PRV[i] in Fs
    if $|CFBF$(f)$| <$ minTableSize or ($|CFBF$(f)$|$ = minTableSize and $|CFBF$(PRV[i])$| > |CFBF$(nextPRV)$|$) then
        minTableSize $\longleftarrow |CFBF$(f)$|$
        nextPRV $\longleftarrow$ PRV[i]
return nextPRV
```

Population Order Heuristic

Another relational heuristic can be motivated by considering our first observation in the examples of section 4.1 which is, eliminating PRVs having variables with large population sizes sooner than others. We call this *population order* heuristic in which we successively branch on the PRV with the smallest CFBF. This heuristic can be implemented in two ways. The first way is to sort the PRVs according to their CFBF at the beginning and use that as the branching order, and the second way is to determine at each round the PRV which has the smallest CFBF among remaining PRVs and branch on it. While the former implementation produces a fixed order, the latter implementation allows for a dynamic ordering (i.e. the algorithm may use different orders in each branch).

Relational Minimum-Fill Heuristic

One possibility for proposing a relational elimination ordering heuristic is to extend min-fill so that it also considers the population sizes. The intuition behind min-fill is that minimizing the number of fill-edges reduces the chance of generating a factor with a large width in the future. Therefore, min-fill weighs all fill-edges equally and successively eliminates a PRV which minimizes the sum of these weights.

In relational models, however, fill-edges can have different sizes. Larger fill-edges can degrade the efficiency more severely than small fill-edges and they should have a higher weight. We consider the weight of each fill-edge to be its table size which is the multiplication of the CFBFs of the PRVs which are connected by the fill-edge. Like min-fill, *relational minimum-fill* heuristic also successively eliminates a PRV which minimizes the sum of these weights. Since each weight is representing the size of a particular table which is separate from other tables, considering the sum of the weights to minimize is reasonable.

Evaluation of Heuristics

In this section, we evaluate and compare our proposed relational heuristics, which are *MinTableSize*, population order and relational min-fill, and also the non-relational min-fill heuristic, which is one of the best known heuristics for non-relational models, based on a series of empirical and theoretical criteria. Experiments were carried out using C++ with Microsoft Visual Studio 2010 on a 2.63GHz core with 2GB of memory under Windows 7.

Empirical Efficiency

To evaluate how efficient each heuristic works, we generated parfactor graphs as follows. First, we generated a random number of PRVs having variables with random populations, and then generated a random number of parfactors. Then, we randomly assigned some of the PRVs to each parfactor. Since all elements of the parfactor graphs that distinguish one graph from another are generated randomly, the whole space of parfactor graphs can be covered in this way and our generated graphs are arguably reasonable representatives to base the experiments on them. The reason for using synthesized graphs for evaluation is that the benchmark graphs used in other works are not big enough (in terms of the number of PRVs and parfactors) to serve as a good representative for evaluating elimination ordering heuristics.

We performed the search-based inference for these parfactor graphs using the elimination ordering heuristics mentioned earlier and recorded the running times of the inference in milliseconds. For each parfactor graph and each heuristic, we ran the program 10 times and computed the average, which is reasonable as the algorithm is deterministic. We set the maximum allowable time for inference to be five minutes (300 seconds). If the program could not find the result in this time, we stopped it and reported "> 300".

Results are presented in Table 1 (the minimum time is in bold). Running time of lifted inference using each of the heuristics are reported in scale of seconds. The first four rows in the table are the parfactor graphs in Examples 2, 3, 4 and 5 respectively and others are parfactor graphs generated as described earlier.

From the table, we can see that *MinTableSize* is outperforming other heuristics in nearly all cases. There are only a few cases where one of the other heuristics has performed better than *MinTableSize*.

Table 1: Running times of lifted inference using different heuristics for elimination ordering

parfactor graphs and population sizes of logical variables	min-fill	relational min-fill	population order	*MinTableSize*
1: {A,B},{B,C(x)},{C(x),A},$\lvert pop(x)\rvert=30$	2.854	2.854	**0.046**	**0.046**
2: {A,B(x)},{A,C(x)},{B(x),C(x)},{B(x),D(x)},{C(x),E(x)},{D(x),E(x)}, $\lvert pop(x)\rvert=5$	79.245	79.245	**0.134**	**0.134**
3: {A(x),B},{A(x),C},{A(x),D},{B,C},{B,D}, $\lvert pop(x)\rvert=25$	2.768	2.768	**0.192**	**0.192**
4: {A,B(x,y)},{B(x,y),C(x,z)},{C(x,z),D}, $\lvert pop(x)\rvert=3$, $\lvert pop(y)\rvert=5$, $\lvert pop(z)\rvert=5$	>300	>300	**0.178**	**0.178**
5: {A,B(x)},{B(x),C(x)}, $\lvert pop(x)\rvert=10$	1.530	1.530	**0.016**	**0.016**
6: {A(x)},{A(x),B(x),C(y)},{A(x),D(y)},{D(y),E,F,G},{D(y),H(z)}, $\lvert pop(x)\rvert=5$, $\lvert pop(y)\rvert=10$, $\lvert pop(z)\rvert=3$	9.617	9.617	>300	**6.618**
7: {A},{A,B},{B,C},{C,D},{C,G},{D,E,G},{A,E,F},{E,H}	**0.460**	**0.460**	0.734	0.567
8: {A(x)},{A(x),B(x),C(y)},{A(x),D(y)},{D(y),E,F,G},{D(y),H(z)}, $\lvert pop(x)\rvert=5$, $\lvert pop(y)\rvert=50$, $\lvert pop(z)\rvert=3$	172.163	172.163	>300	**104.751**
9: {A,B(x)},{B(x),C(x)},{D(x),E(x)},{E(x),F(x)},{G(x)}, $\lvert pop(x)\rvert=10$	1.555	1.555	**0.062**	**0.062**
10: {A(x),B(y)},{A(x),C(z)},{B(y),D(x)},{C(z),D(x)}, $\lvert pop(x)\rvert=5$, $\lvert pop(y)\rvert=10$, $\lvert pop(z)\rvert=15$	69.410	29.400	22.157	**21.919**
11: {A(x),B(y)},{A(x),E},{D(x,y),E},{C,D(x,y)},{C,B(y)},$\lvert pop(x)\rvert=7$, $\lvert pop(y)\rvert=18$	>300	**0.746**	**0.746**	**0.746**
12: {A,B(x),E(x)},{C(x),D(y),B(x)},{E(x),C(x)},{C(x),B(x)},{A,D(y)}, $\lvert pop(x)\rvert=2$, $\lvert pop(y)\rvert=5$	17.785	2.420	2.445	**2.409**
13: {A(x),B(y)},{A(x),C(y)},{A(x),D(y)},{E(y),B(y)},{E(y),C(y)},{E(y),D(y)}, $\lvert pop(x)\rvert=20$, $\lvert pop(y)\rvert=2$	14.678	**6.245**	10.026	**6.245**
14: {A(x),B(y),C(y)},{A(x),D(y)},{E(x),B(y),C(y)},{E(y),D(y)}, $\lvert pop(x)\rvert=15$, $\lvert pop(y)\rvert=3$	>300	192.925	**34.411**	34.710
15: {A(x),B(y),C(x)},{B(y),C(x),F(z)},{A(x),F(z),D(w)},{B(y),E(w)},{C(x),D(w)},$\lvert pop(x)\rvert=2$,$\lvert pop(y)\rvert=5$,$\lvert pop(z)\rvert=10$,$\lvert pop(w)\rvert=30$	256.192	256.192	56.880	**56.274**
16: {A(x),B(x,y)},{B(x,y),C(y)},{A(x),D,E},{C(y),F}, $\lvert pop(x)\rvert=6$, $\lvert pop(y)\rvert=16$	>300	>300	1.273	**0.926**
17: {A(x),B(y)},{B(y),E,F},{A(x),C,D}, $\lvert pop(x)\rvert=5$, $\lvert pop(y)\rvert=10$	**0.882**	**0.882**	1.886	**0.882**
18: {A(x),B(x,y)},{B(x,y),C(x,y),D(y)},{A(x),D(y)}, $\lvert pop(x)\rvert=4$, $\lvert pop(y)\rvert=7$	183.353	183.353	**0.220**	0.540

Furthermore, we can see that the population order heuristic results in a better running time than min-fill and relational min-fill in most cases. However, it has a very bad performance on some parfactor graphs. The reason is that population order heuristic does not take into account how big the newly generated parfactors are.

We can also see that min-fill and relational min-fill produce the same results in most cases. That's mostly because in some parfactor graphs, we can successively eliminate a PRV which produces no fill-edges, or the generated fill-edges have the same CFBFs. In these cases, both min-fill and relational min-fill choose the same PRV to be eliminated and have the same performance. However, in cases where they produced different results, relational min-fill has a better performance than min-fill.

Adaptation to Population Change

In relational models, varying population sizes are quite common (see some examples in (Kazemi et al. 2014)). To see how each heuristic performs as the population sizes of the logical variables change, we used four parfactor graphs of Table 1 (6th, 10th, 12th and 13th parfactor graphs) and varied the population size of one of the logical variables to see how the running time of lifted search-based inference with different heuristics is affected. These four parfactor graphs were chosen because different orders are suggested by different heuristics in most cases, and they have a logical variable where inference performance is highly dependent on its population size. Results are shown in Figure 3 where the x-axis shows the population size of the logical variable being changed and the y-axis shows time in seconds. We can see that *MinTableSize* is doing a better job of finding a new efficient elimination order as we change the population sizes of logical variables.

Avoiding Grounding Populations

The *MinTableSize* heuristic avoids unnecessary grounding of a population. We can prove the following proposition:

Proposition If any order results in not grounding any population, then *MinTableSize* will produce an order that results in not grounding any population.

Proof. MinTableSize branches firstly on PRVs with fewer than two logical variables. Branching on these PRVs does not require grounding any population. After branching on these PRVs, if there exists an order that results in not grounding any population, there should be a PRV having zero or one logical variables (otherwise branching on any of the remaining PRVs needs grounding). This means that at least one of the logical variables has been replaced by a constant. We know this happens when all PRVs of all parfactors have the same logical variable, so at least one of the logical variables has been replaced by a constant in all remaining PRVs. Since all PRVs have the same number of logical variables replaced by a constant, those with fewer logical variables at the beginning still have fewer logical variables. Therefore, the PRV with minimum number of logical variables (which should be chosen next to avoid grounding) is now the one that used to have the minimum number of logical variables at the beginning of the algorithm among those with at least two logical variables. This PRV is exactly the one chosen by *MinTableSize*, because *MinTableSize* sorts all PRVs with at least two logical variables ascending for branching on.

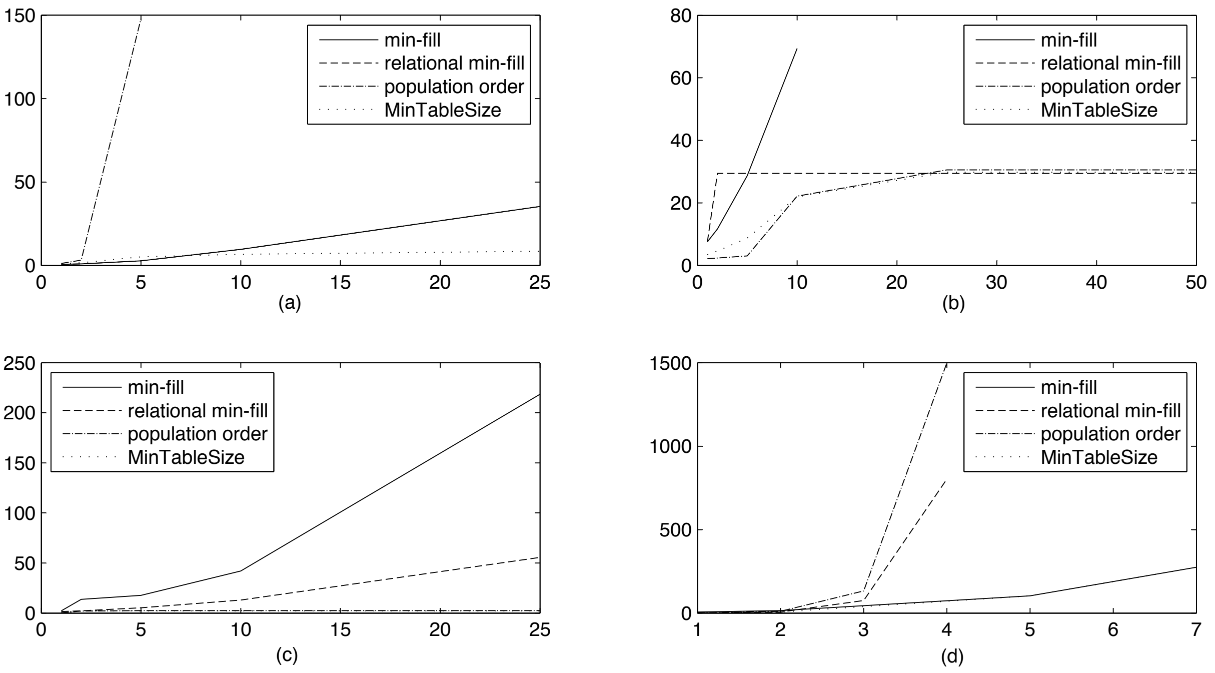

Figure 3: Running time of lifted search-based inference as a function of population size for different heuristics. From Table 1, these are (a) 6th parfactor graph, (b) 10th parfactor graph, (c) 12th parfactor graph, (d) 13th parfactor graph. In all cases the x-axis is the population of *y*. The *y*-axis is runtime in milliseconds. In (a) min-fill and relational min-fill are overlapping. In (c) *MinTableSize* and population order heuristics are overlapping. In (d) *MinTableSize* and min-fill are overlapping.

The analysis for PRVs with more logical variables is the same. □

Even though the intuition behind population order heuristic helps avoiding to ground populations, the above proposition does not hold for this heuristic:

Example 6. Consider a parfactor graph with only one parfactor $\{A(x), B(y,z)\}$ and suppose $|pop(x)| = 3000$, $|pop(y)| = 50$ and $|pop(z)| = 50$. Population order heuristic selects B as the first PRV to branch on because it has a smaller CFBF than A. Since B has more than one logical variable, the search-based inference algorithm grounds one of the populations. This is a case when avoiding to ground a population is possible but min-fill and relational min-fill end up grounding a population.

Computational Complexity

Suppose there are n PRVs. According to Algorithm 1, in order to find the next PRV to eliminate using *MinTableSize* heuristic, we have to loop over all PRVs twice. We have to do this process n times to find the total order. This makes the time complexity of *MinTableSize* $O(n^2)$.

For the population order heuristic, we proposed two different implementations. The time complexity of the first one, which is sorting the PRVs according to their CFBFs at the beginning, is the same as the time complexity of sorting which is $O(n\log(n))$. The other implementation, which determines the PRV with minimum CFBF at each step and branches on it, does an $O(n)$ process (finding the PRV with minimum CFBF) n times. This gives a time complexity of $O(n^2)$.

To find the next PRV to eliminate using min-fill or relational min-fill, we have to loop over all PRVs. In the i-th iteration, we consider the parfactor f resulting from eliminating i-th PRV. Then we should have two nested loops over the PRVs in f to count how many couples of PRVs in f have not been already connected. This makes the time complexity of finding the next PRV to be $O(n^3)$. Since we do this n times to find the total order, the time complexity of min-fill and relational min-fill is $O(n^4)$, although enhancements are possible for certain models (e.g. sparse graphs) by using (in most cases) more memory (e.g. see (Kask et al. 2011)).

Conclusion

We examined the importance of elimination ordering in lifted first-order probabilistic inference and proposed three heuristics for that. We showed empirically that the *MinTableSize* heuristic outperforms non-relational min-fill, which is widely used for non-relational models, relational min-fill and population order heuristics and also does a better job of adapting itself to the changes in the population sizes of logical variables. We presented an $O(n^2)$ algorithm, where n represents the number of PRVs in the model, for this heuristic and proved that it will produce an order that results in not grounding any population if there exists at least one such order.

Acknowledgements

This work was supported by NSERC. Thanks to David Buchman for valuable feedback.

References

Arnborg, S.; Corneil, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods* 8(2):277–284.

Berry, A.; Blair, J. R.; Heggernes, P.; and Peyton, B. W. 2004. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica* 39(4):287–298.

Clautiaux, F.; Ngre, A. M. S.; and Carlier, J. 2004. Heuristic and metaheuristic methods for computing graph treewidth. *RAIRO-Operations Research* 38(1):13–26.

Cooper, G. F. 1990. The computational complexity of probabilistic inference using Bayesian networks. *Artificial Intelligence* 42(2-3):393–405.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 126(1-2):5–41.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. *Lifted first-order probabilistic inference. In L. Getoor and B. Taskar (Eds).* MIT Press.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann, San Fransisco, CA.

den Broeck, G. V.; Taghipour, N.; Meert, W.; Davis, J.; and Raedt, L. D. 2011. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of International Joint Conference on AI (IJCAI)*, 2178–2185.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA.

Gogate, V., and Domingos, P. 2010. Exploiting logical structure in lifted probabilistic inference. In *Proceedings of Statistical Relational Artificial Intelligence*.

Jha, A.; Gogate, V.; Meliou, A.; and Suciu, D. 2010. Lifted inference from the other side: The tractable features. *Twenty-Fourth Annual Conference on Neural Information Processing Systems (NIPS).*

Kask, K.; Gelfand, A.; Otten, L.; and Dechter, R. 2011. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *AAAI*.

Kazemi, S. M.; Buchman, D.; Kersting, K.; Natarajan, S.; and Poole, D. 2014. Relational logistic regression. In *Proc. 14th International Conference on Principles of Knowledge Representation and Reasoning (KR).*

Kjaerulff, U. 1985. Triangulation of graphs–algorithms giving small total state space. Technical report, Deptartment of Mathematics and Computer Science, Aalborg University, Denmark.

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

Larranaga, P.; Kuijpers, C. M.; Poza, M.; and Murga, R. H. 1997. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing* 7(1):19–34.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaelbling, L. P. 2008. Lifted probabilistic inference with counting formulae. In *Proceedings of the Twenty Third Conference on Advances in Artificial Intelligence (AAAI)*, 1062–1068.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaumann.

Poole, D.; Bacchus, F.; and Kisynski, J. 2011. Towards completely lifted search-based probabilistic inference. *arXiv:1107.4035 [cs.AI].*

Poole, D. 2003. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 985–991.

Rose, D. J.; Tarjan, R. E.; and Lueker, G. S. 1976. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing* 5(2):266–283.

Sato, N., and Tinney, W. F. 1963. Techniques for exploiting the sparsity of the network admittance matrix. *Power Apparatus and Systems, IEEE Transactions on* 82(69):944–950.

Tarjan, R. E., and Mihalis, Y. 1984. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing* 13(3):566–579.

Zhang, N. L., and Poole, D. 1994. A simple approach to Bayesian network computations. In *Proceedings of the 10th Canadian Conference on AI*, 171–178.