k-Optimal: A Novel Approximate Inference Algorithm for ProbLog

Joris Renkens, Guy Van den Broeck, and Siegfried Nijssen

joris.renkens@student.kuleuven.be
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Abstract. ProbLog is a probabilistic extension of Prolog. Given the complexity of exact inference under ProbLog's semantics, in many applications approximate inference is necessary. Current approximate inference algorithms for ProbLog however require either dealing with large numbers of proofs or do not guarantee a low approximation error. In this paper we introduce a new approximate inference algorithm which addresses these shortcomings. Given a user-specified parameter k, this algorithm approximates the success probability of a query based on at most k proofs and ensures that the calculated probability p is $(1 - 1/e)p^* \leq p \leq p^*$, where p^* is the highest probability that can be calculated based on any set of k proofs. Furthermore a useful feature of the set of calculated proofs is that it is diverse. We show that this type of inference is in particular useful in solving decision problems.

1 Introduction

ProbLog [7] is a probabilistic extension of Prolog. It has been used to solve learning problems in probabilistic networks as well as other types of probabilistic data [8]. The key feature of ProbLog is its distribution semantics. Each fact in a ProbLog program can be annotated with the probability that this fact is true in a random sample from the program. The success probability of a query is equal to the probability that the query succeeds in a sample from the program, where facts are sampled independently from each other. Each such sample is also called a *possible world*.

The main problem in calculating the success probability of a query in ProbLog is the high computational complexity of exact inference. As multiple proofs for a query can be true in a possible world, we cannot calculate the success probability of a query based on the probabilities of the independent proofs; we need to deal with a *disjoint sum problem* [7]. This problem becomes worse as the number of proofs grows.

To deal with this computational issue, several approaches have been proposed in the past. In [7] it was proposed to use Binary Decision Diagrams (BDDs) [2] to deal with the disjoint sum problem. BDDs can be seen as a representation of

proofs from which the required success probability can be calculated in polynomial time. Building a BDD for all proofs can however be intractable. In [7] it was shown that for a given desired approximation factor ϵ, an iterative deepening algorithm can be used to approximate the success probability from a subset of proofs. However, to reach reasonable approximation errors in practice, this algorithm still needs to compile large numbers of proofs into a BDD [7].

A commonly used alternative which does not have this disadvantage is the *k-best* strategy. In this case the k most likely proofs are searched for, where k is a user-specified parameter; a BDD is constructed based on these proofs only. Whereas this strategy avoids compiling many proofs, its disadvantage is that one has few guarantees with respect to the quality of the calculated probability: it is not clear whether any other set of k proofs would achieve a better approximation, or how far the calculated probability is from its true value.

A further disadvantage of k-best is found in the use of ProbLog in solving probabilistic decision problems [1]. An example of such decision problem is the problem of targeted advertising in social networks: one wishes to identify a small subset of nodes in a social network such that the expected number of people reached is maximized. Such problems can be solved by defining an optimization problem on top of a set of BDDs. To ensure that the calculation remains tractable it is important that these BDDs are as small as possible while still carrying all relevant information. As we will show, the k-best strategy selects a set of proofs that is not optimal for this purpose. Intuitively the reason is that the k-best proofs can be highly redundant with respect to each other.

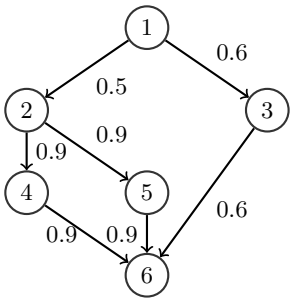
In this paper we propose a new algorithm, k-optimal, for finding a set of at most k proofs. The key distinguishing feature with respect to k-best is that it ensures that the set of k proofs found is of provably good quality. In particular, if p^* is the best probability that can be calculated based on k proofs, our algorithm will not calculate a probability that is worse than $(1-1/e)p^*$. At the same time, our algorithm ensures that the resulting set of proofs is diverse, i.e., proofs are less likely to use similar facts; the resulting set of proofs carries more of the probability mass of the exact success probability.

The remainder of this paper is organized as follows: Section 2 introduces ProbLog and discusses the drawbacks of k-best; Section 3 introduces the new algorithm; Section 4 proves the quality of the resulting set of proofs; Section 5 reports on some experiments and finally Section 6 concludes.

2 ProbLog and *k*-Best

In Problog a program consists of two parts: a collection of probabilistic facts and a collection of rules. The rules are written in standard Prolog syntax. The syntax for the probabilistic facts is $p_i :: f_i$. Here p_i denotes the probability of the fact f_i.

Example 1. This running example shows the implementation of the small probabilistic network of Figure 1. Besides the network, the definition of a path in the network is implemented.




```
0.5::edge(1,2).
0.6::edge(1,3).
0.9::edge(2,4).
0.9::edge(2,5).
0.6::edge(3,6).
0.9::edge(4,6).
0.9::edge(5,6).

path(X,X).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

Fig. 1. A small probabilistic network and its ProbLog representation

The success probability of a query is calculated from the BDD representation of its proofs. In our network, each proof reflects one set of edges that needs to be present in a possible world in order for a path to exist. Worlds in which a path between node 1 and 6 exists are hence characterized by the following formula in disjunctive normal form (DNF):

$$(e(1,2) \wedge e(2,4) \wedge e(4,6)) \vee (e(1,2) \wedge e(2,5) \wedge e(5,6)) \vee (e(1,3) \wedge e(3,6)) \quad (1)$$

Here the fact $edge(X,Y)$ is represented by $e(X,Y)$. In ProbLog this formula is transformed into its BDD representation. A dynamic programming algorithm allows to calculate the success probability from this BDD efficiently.

In cases in which building the BDD from all proofs is intractable, k-best [4] approximates the success probability using only the k most probable proofs. We show on our example that this is not always a good solution. We approximate the probability of the query $path(1,6)$ in Figure 1 by using only the two most probable proofs. In this case the proofs $p_1 = e(1,2) \wedge e(2,4) \wedge e(4,6)$ and $p_2 = e(1,2) \wedge e(2,5) \wedge e(5,6)$ are selected. They both have a probability equal to $0.5 \cdot 0.9 \cdot 0.9 = 0.405$, while the third proof p_3 has a probability equal to $0.6 \cdot 0.6 = 0.36$. Based on these two proofs $\mathrm{P}(path(1,6)) = 0.5 \times (1 - (1 - 0.9 \times 0.9)^2) = 0.48195$.

Avoiding redundancy in the proofs can lead to better results. When the two proofs p_1 and p_3 are used, we get a much higher success probability, $\mathrm{P}(path(1,6)) = \mathrm{P}(p_1 \vee p_3) = 0.6192$. This result is achieved even though the probability of p_3 is lower than the one of proof p_2, because there is a large overlap between the possible worlds in which p_1 and p_2 are true.

3 *k*-Optimal

The problem that our algorithm addresses is the following.

Given the collection of all possible proofs for a goal (V) and a maximum number of proofs which can be used (k).
Find a collection $A = \arg\max_{B \subseteq V, |B| \leq k} \mathrm{P}(\bigvee_{p \in B} p)$.

The algorithm that we propose is a simple greedy algorithm in which in each iteration the proof is added that increases the probability most, as given below.

Clearly, the main task that needs to be addressed is the efficient calculation of $\arg\max_{p \in V} \mathrm{P}(A \cup \{p\})$. Let us first consider the problem of evaluating $\mathrm{P}(A \cup \{p\})$ for a particular proof p.

Algorithm 1 *greedy_solve*(V)

$A = \emptyset$
for $i = 1..k$ **do**
  $A = A \cup \arg\max_{p \in V} \mathrm{P}(A \cup \{p\})$
end for
return A

Calculating $\mathrm{P}(A \cup \{p\})$. In a naïve approach this would involve building the BDD for $A \cup \{p\}$ from scratch for each p. Fortunately, we can avoid this. Let *dnf* represent the DNF formula $\bigvee_{p' \in A} p'$ for the set of proofs A and let $p \equiv \bigwedge_i f_i$. Then $\mathrm{P}(f_1 \wedge \cdots \wedge f_n \vee \mathit{dnf}) = \mathrm{P}(f_1 \wedge \cdots \wedge f_n) + \mathrm{P}(\neg f_1 \wedge \mathit{dnf}) + \mathrm{P}(f_1 \wedge \neg f_2 \wedge \mathit{dnf}) + \cdots + \mathrm{P}(f_1 \wedge \cdots \wedge f_{n-1} \wedge \neg f_n \wedge \mathit{dnf})$. A term in this sum can be calculated as follows:

$$\mathrm{P}(f_1 \wedge \cdots \wedge f_{i-1} \wedge \neg f_i \wedge \mathit{dnf}) = \mathrm{P}(f_1) \ldots \mathrm{P}(f_{i-1})(1 - \mathrm{P}(f_i))\, \mathrm{P}(\mathit{dnf} | f_1 \wedge \cdots \wedge f_{i-1} \wedge \neg f_i),$$

where we still need to specify how to calculate $\mathrm{P}(\mathit{dnf} | f_1 \wedge \cdots \wedge f_{i-1} \wedge \neg f_i)$. Fortunately, this term can be calculated efficiently if we already have the BDD for the *dnf* formula. We reuse the algorithm of [7] for calculating $\mathrm{P}(\mathit{dnf})$ from a BDD, where during the traversal of the BDD, we assume that $\mathrm{P}(f_j) = 1$ for the conditional facts $j < i$ while setting $\mathrm{P}(f_i) = 0$ for the other facts.

Overall, our strategy is hence to compile the BDD for the set of proofs A at the beginning of each iteration of the for-loop; when calculating the probability of a set $A \cup \{p\}$, we traverse this BDD once, calculating n terms for each node of the BDD.

This computation can be optimized further by reusing computations; furthermore, facts in the proof which are not represented in the DNF can be ignored. Overall, this means that we do not always need to traverse the BDD. The details of these optimizations are omitted here.

Finding a proof. In each iteration we need to search for the proof that maximizes $\mathrm{P}(A \cup \{p\})$. To find such a proof without enumerating all proofs in V we use a branch-and-bound search similar to that of k-best [7]. When a proof is found, we calculate $b = \mathrm{P}(A \cup \{p\}) - \mathrm{P}(A)$. We prune a proof $\{f_1, \ldots, f_n\}$ if the individual probability $\mathrm{P}(f_1) \ldots \mathrm{P}(f_n) \leq b^*$, where b^* is the best b value seen till that moment. Here we exploit that $\arg\max_{p \in V} \mathrm{P}(A \cup \{p\}) = \arg\max_{p \in V} (\mathrm{P}(A \cup \{p\}) - \mathrm{P}(A))$, while $(\mathrm{P}(A \cup \{p\}) - \mathrm{P}(A)) \leq \mathrm{P}(f_1) \ldots \mathrm{P}(f_n)$.

4 Analysis

The quality of the result of our algorithm follows from the fact that the function $\mathrm{P}(.)$ is *submodular* and *monotone* [3, 5].

Definition 1 (Submodular Function). *A function f is submodular when* $\forall A \subseteq B \subseteq S, \forall x \in S : f(A \cup x) - f(A) \geq f(B \cup x) - f(B)$.

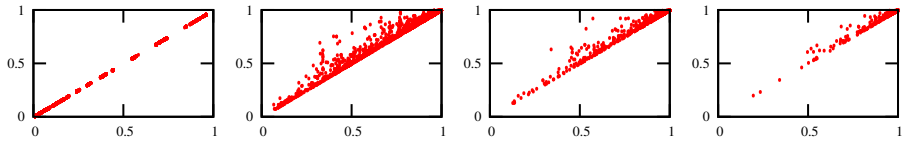


Fig. 2. Ratio of probabilities by k-optimal and k-best, for k equal to 1, 7, 13 and 20

Definition 2 (Monotone Function). *A function f is monotone when* $\forall A, B \subseteq S : A \subseteq B \rightarrow f(A) \leq f(B)$.

Clearly, adding a proof to a larger set of proofs will decrease its impact on the overall probability: as more possible worlds will already be covered, a larger set of proofs will result in a larger probability.

It was shown in [3] that for submodular and monotonic functions the iterative greedy algorithm that we employ finds a solution for which the function value is at least $1 - \frac{1}{e}$ times the one of the optimal solution.

5 Experiments

Three types of experiments have been executed. In the first experiment, the results of k-optimal are compared with those of k-best. For this purpose, the probabilistic network constructed in [6] is used. This network contains 15147 bidirectional edges, 5568 unidirectional edges and 5313 nodes. Furthermore 5371 pairs of nodes are given. For these pairs, k-best and k-optimal are used to calculate the success-probability of a path between the pairs of nodes with a maximal length of four. In Figure 2 the probabilities are shown for $k = 1$, $k = 7$, $k = 13$ and $k = 20$. The x-value is equal to the probability achieved with k-best. The y-value is equal to the probability achieved with k-optimal. Only the pairs are shown which have at least $k + 1$ proofs. When this is not the case, all the proofs are selected and k-best and k-optimal achieve the same result.

When k is equal to one, there is no difference between k-best and k-optimal. In the other cases, k-optimal performs better than k-best. When k is low compared to the number of available proofs ($k = 7$) k-optimal achieves better results. Run time experiments (not shown) reveal that k-optimal is slower than k-best.

In the second experiment we evaluate a modified k-optimal algorithm (k-Θ-optimal), in which the iterative loop is stopped early if no single proof can be found that improves the probability with a user-specified threshold θ. The main motivation is to reduce the size of the set of proofs further. Results are shown in Figure 3, in which we plot the average probability in function of the average number of used proofs. These numbers are calculated for $k \in \{1, \ldots, 20\}$. k-Θ-Optimal uses fewer proofs to calculate the probability. This does not result in a large drop of the calculated success probability. We can conclude that k-Θ-Optimal successfully avoids adding insignificant proofs.

The third experiment (also Figure 3) evaluates k-optimal and k-Θ-optimal on a DTProbLog decision problem. In this problem we aim to select a subset of n edges such that the expected number of connected pairs of nodes in a given set is

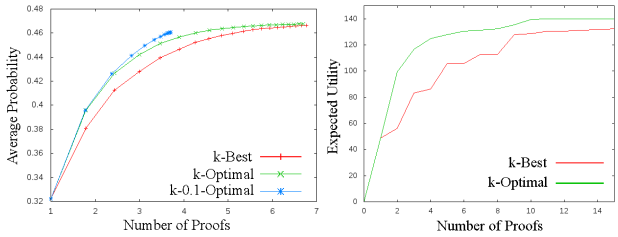


Fig. 3. Average utility and probability in function of the average number of used proofs

maximized [8]. We evaluate the solutions by calculating exact path probabilities on the selected subnetworks. k-optimal finds a better solution.

6 Conclusions

We have introduced a new approximating inference-mechanism to calculate the success-probability of a query in ProbLog. This mechanism uses k proofs to approximate the exact probability. As k-optimal searches for a proof that increases the probability most, it minimizes the redundancy between the selected proofs. An efficient calculation of this probability was proposed. In contrast to k-best we could also show that it finds a proof set that is close to optimal for its size.

We presented an extension of k-optimal which avoids adding insignificant proofs. This is easily possible in k-optimal as k-optimal computes the added probability of a proof. k-Θ-Optimal produces fewer proofs, with only a small loss of probability.

References

1. G. Van den Broeck, I. Thon, M. van Otterlo, and L. De Raedt. *DTProbLog: A Decision-Theoretic Probabilistic Prolog.* AAAI, 2010.
2. R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.
3. G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. *Management Science*, 1977
4. A. Kimmig, V. Santos Costa, R. Rocha, B. Demoen, and L. De Raedt. On the efficient execution of problog programs. ICLP, pages 175–189, 2008.
5. G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
6. O. Ourfali, T. Shlomi, T. Ideker, E. Ruppin, and R. Sharan. Spine: a frame- work for signaling-regulatory pathway inference from cause-effect experiments. *Bioinformatics*, 23(13):359–366, 2007.
7. L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. *IJCAI*, pages 2462–2467, 2007.
8. L. De Raedt, A. Kimmig, B. Gutmann, K. Kersting, V. Santos Costa, and H. Toivonen. Probabilistic inductive querying using ProbLog. *Inductive Databases and Constraint-Based Data Mining*, pages 229–262, 2010.
9. C.-H. Yeang, T. Ideker, and T. Jaakkola. Physical network models. *Journal of Computational Biology*, 11(2/3):243–262, 2004.