DTPROBLOG: A Decision-Theoretic Probabilistic Prolog

Guy Van den Broeck and **Ingo Thon** and **Martijn van Otterlo** and **Luc De Raedt**

Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
{guy.vandenbroeck, ingo.thon, martijn.vanotterlo, luc.deraedt}@cs.kuleuven.be

Abstract

We introduce DTPROBLOG, a decision-theoretic extension of Prolog and its probabilistic variant ProbLog. DTPROBLOG is a simple but expressive probabilistic programming language that allows the modeling of a wide variety of domains, such as viral marketing. In DTPROBLOG, the utility of a strategy (a particular choice of actions) is defined as the expected reward for its execution in the presence of probabilistic effects. The key contribution of this paper is the introduction of exact, as well as approximate, solvers to compute the optimal strategy for a DTPROBLOG program and the decision problem it represents, by making use of binary and algebraic decision diagrams. We also report on experimental results that show the effectiveness and the practical usefulness of the approach.

1 Introduction

Artificial intelligence is often viewed as the study of how to act rationally (Russell and Norvig 2003). The problem of acting rationally has been formalized within decision theory using the notion of a *decision problem*. In this type of problem, one has to choose actions from a set of alternatives, given a utility function. The goal is to select the strategy (set or sequence of actions) that maximizes the utility function. While the field of decision theory has devoted a lot of effort to deal with various forms of knowledge and uncertainty, there are so far only a few approaches that are able to cope with both uncertainty and rich logical or relational representations (see (Poole 1997; Nath and Domingos 2009; Chen and Muggleton 2009)). This is surprising, given the popularity of such representations in the field of statistical relational learning (Getoor and Taskar 2007; De Raedt et al. 2008).

To alleviate this situation, we introduce a novel framework combining ProbLog (De Raedt, Kimmig, and Toivonen 2007; Kimmig et al. 2008), a simple probabilistic Prolog, with elements of decision theory. The resulting probabilistic programming language DTPROBLOG (Decision-Theoretic ProbLog) is able to elegantly represent decision problems in complex relational and uncertain environments. A DTPROBLOG program consists of a set of definite clauses (as in Prolog), a set of probabilistic facts (as in ProbLog), and, in addition, a set of decision facts, specifying which decisions are to be made, and a set of utility attributes, specifying the rewards that can be obtained. Further key contributions of this paper include the introduction of an exact algorithm for computing the optimal strategy as well as a scalable approximation algorithm that can tackle large decision problems. These algorithms adapt the BDD based inference mechanism of ProbLog. While DTProbLog's representation and spirit are related to those of e.g. (Poole 1997) for ICL, (Chen and Muggleton 2009) for SLPs, and (Nath and Domingos 2009) for MLNs, its inference mechanism is distinct in that it employs state-of-the-art techniques using decision diagrams for computing the optimal strategy exactly; cf. the related work section for a more detailed comparison.

The paper is organized as follows: in Section 2, we introduce DTPROBLOG and its semantics; Section 3 discusses inference and Section 4 how to find the optimal strategy for a DTPROBLOG program; Section 5 reports on some experiments and Section 6 describes related work; Finally, we conclude in Section 7. We assume familiarity with standard concepts from logic programming (see e.g. Flach (1994)).

2 Decision-Theoretic ProbLog

ProbLog (De Raedt, Kimmig, and Toivonen 2007; Kimmig et al. 2008) is a recent probabilistic extension of Prolog. A ProbLog theory $\mathcal{T}$ consists of a set of labeled facts $\mathcal{F}$ and a set of definite clauses $\mathcal{BK}$ that express the background knowledge. The facts $p_i :: f_i$ in $\mathcal{F}$ are annotated with a probability p_i stating that $f_i\theta$ is true with probability p_i for all substitutions θ grounding f_i. These random variables are assumed to be mutually independent. A ProbLog theory describes a probability distribution over Prolog programs $L = \mathcal{F}_L \cup \mathcal{BK}$ where $\mathcal{F}_L \subseteq \mathcal{F}\Theta$ and $\mathcal{F}\Theta$ denotes the set of all possible ground instances of facts in $\mathcal{F}$.[1]

$$\mathrm{P}(L|\mathcal{T}) = \prod_{f_i \in \mathcal{F}_L} p_i \prod_{f_i \in \mathcal{F}\Theta \setminus \mathcal{F}_L} (1 - p_i)$$

The *success probability* of a query q is then

$$\mathrm{P}(q|\mathcal{T}) = \sum_L \mathrm{P}(q|L) \cdot \mathrm{P}(L|\mathcal{T}) \qquad (1)$$

[1] Throughout the paper, we shall assume that $\mathcal{F}\Theta$ is finite for notational convenience, but see (Sato 1995) for the infinite case.

where $\mathrm{P}(q|L) = 1$ if there exists a θ such that $L \models q\theta$.

Observe also that ProbLog defines a probability distribution P_w over possible worlds, that is, Herbrand interpretations. Indeed, each atomic choice, or each $\mathcal{F}_L \subseteq \mathcal{F}\Theta$, can be extended into a possible world by computing the least Herbrand model of $L = \mathcal{F}_L \cup \mathcal{BK}$. This possible world is assigned the probability $P_w = \mathrm{P}(L|\mathcal{T})$.

In addition to the background knowledge $\mathcal{BK}$ and the probabilistic facts $\mathcal{F}$ with their probabilities, a DTPROBLOG program consists of a set of decision facts $\mathcal{D}$ and utility attributes $\mathcal{U}$, which we will now define.

Decisions and Strategies

Decision variables are represented by facts, and so, by analogy with the set of probabilistic facts $\mathcal{F}$, we introduce $\mathcal{D}$, the set of decision facts of the form $? :: d$. Each such d is an atom and the label $?$ indicates that d is a decision fact. Note that decision facts can be non-ground.

A *strategy* σ is then a function $\mathcal{D} \rightarrow [0, 1]$, mapping a decision fact to the probability that the agent assigns to it. Observe that there is one probability assigned to each decision fact. All instances – these are the groundings – of the same decision fact are assigned the same probability, realizing parameter tying at the level of decisions. For a set of decision facts $\mathcal{D}$ we denote with $\sigma(\mathcal{D})$ the set of probabilistic facts obtained by labeling each decision fact $(? :: d) \in \mathcal{D}$ as $\sigma(d) :: d$. Taking into account the decision facts $\mathcal{D}$ and the strategy σ, the success probability can be defined as:

$$\mathrm{P}(q|\mathcal{F} \cup \sigma(\mathcal{D}), BK),$$

which, after the mapping $\sigma(\mathcal{D})$, is a standard ProbLog query. Abusing notation we will use $\sigma(\mathcal{DT})$ to denote ProbLog program $\sigma(D) \cup \mathcal{F}$ with the background knowledge $\mathcal{BK}$.

Example 1. As a running example we will use the following problem of dressing for unpredictable weather:

$\mathcal{D} = \{? :: \texttt{umbrella}, ? :: \texttt{raincoat}\}$ $\quad$ $\mathcal{F} = \{0.3 :: \texttt{rainy}, 0.5 :: \texttt{windy}\}$

$\mathcal{BK} =$
```
broken_umbrella :- umbrella, rainy, windy.
dry :- rainy, umbrella, not(broken_umbrella).
dry :- rainy, raincoat.
dry :- not(rainy).
```

There are two decisions to be made: whether to bring an umbrella and whether to wear a raincoat. Furthermore, `rainy` and `windy` are probabilistic facts. The background knowledge describes when one gets wet and when one breaks the umbrella due to heavy wind. The probability of `dry` for the strategy $\{\texttt{umbrella} \mapsto 1, \texttt{raincoat} \mapsto 1\}$ is 1.0.

Rewards and Expected Utility

The set $\mathcal{U}$ consists of utility attributes of the form $u_i \rightarrow r_i$, where u_i is a literal and r_i a reward for achieving u_i. The semantics is that whenever the query u_i succeeds, this yields a reward of r_i. Thus, utility attributes play a role analog to queries in Pro(b)log. The reward is given only once, regardless for how many substitutions it succeeds.

For a Prolog program L, defining a possible world through its least Herbrand model, we define the *utility* of u_i to be the reward due to u_i. The utility attributes have to be additive such that

$$\mathrm{Util}(L) = \sum_{(u_i \rightarrow r_i) \in \mathcal{U}} r_i \cdot \mathrm{P}(u_i|L).$$

Because a ProbLog theory $\mathcal{T}$ defines a probability distribution over Prolog programs, this gives a total expected utility of

$$\mathrm{Util}(\mathcal{T}) = \sum_{(u_i \rightarrow r_i) \in \mathcal{U}} r_i \cdot \mathrm{P}(u_i|\mathcal{T})$$

where $P(u_i|\mathcal{T})$ is defined in Equation 1.

Example 2. We extend Example 1 with utilities:

`umbrella` $\rightarrow -2$ $\quad$ `dry` $\rightarrow 60$

`raincoat` $\rightarrow -20$ $\quad$ `broken_umbrella` $\rightarrow -40$

Bringing an umbrella, breaking the umbrella or wearing a raincoat incurs a cost. Staying dry gives a reward.

Given these definitions, the semantics of a DTPROBLOG theory is defined in terms of the utility of a strategy σ. The expected utility of a single utility attribute $a_i = (u_i \rightarrow r_i)$ is

$$\mathrm{Util}(a_i|\sigma, \mathcal{DT}) = r_i \cdot P(u_i|\sigma(\mathcal{DT})) \quad (2)$$

and the total utility is

$$\mathrm{Util}(\sigma, \mathcal{DT}) = \sum_{a_i \in \mathcal{U}} \mathrm{Util}(a_i|\sigma(\mathcal{DT})) = \mathrm{Util}(\sigma(\mathcal{DT})). \quad (3)$$

Thus, the total utility is the sum of the utilities of each utility attribute and the expected utility for a single attribute is proportionate to its success probability.

3 Inference

The first problem that we will tackle is how to perform inference in DTPROBLOG, that is, how to compute the utility $Util(\sigma, \mathcal{DT})$ of a particular strategy σ in a DTPROBLOG program $\mathcal{DT}$. This is realized by first computing the success probability of all utility literals u_i occurring in $\mathcal{U}$ using the standard ProbLog inference mechanism. The overall utility $Util(\sigma, \mathcal{DT})$ can then be computed using Eq. 3. Let us therefore sketch how ProbLog answers the queries $\mathrm{P}(q|\sigma(\mathcal{DT}))$. This is realized in two steps; cf. (De Raedt, Kimmig, and Toivonen 2007; Kimmig et al. 2008) .

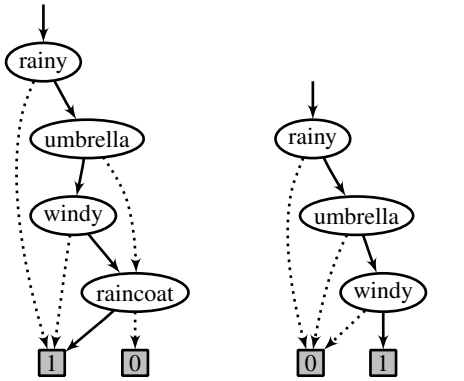
First, all different proofs for the query q are found using SLD-resolution in the ProbLog program $\sigma(\mathcal{DT})$. The probabilistic facts and decisions that are used in these proofs are gathered in a DNF formula. Each proof relies on the conjunction of probabilistic facts that needs to be true to prove q and, hence, the DNF formula represents the disjunction of these conditions. This reduces the problem of computing the probability of the query q to that of computing the probability of the DNF formula. However, because the conjunctions in the different proofs are not mutually exclusive, one cannot compute the probability of the DNF formula as the sum of the probabilities of the different conjunctions as this would lead to values larger than one. Therefore, the second step

Algorithm 1 Calculating the probability of a BDD

function PROB(BDD-node n)
  if n is the 1-terminal **then return** 1
  if n is the 0-terminal **then return** 0
  let h and l be the high and low children of n
  return $p_n \cdot \text{PROB}(h) + (1 - p_n) \cdot \text{PROB}(l)$

solves this disjoint-sum problem by constructing a Binary Decision Diagram (BDD) (Bryant 1986) that represents the DNF formula. A BDD is an efficient graphical representation for boolean formulas. Example BDDs are shown in Figure 1. The BDD can be seen as a decision tree. To determine whether a boolean variable assignment satisfies the formula represented by the BDD, one starts at the root of the BDD and depending on the value of the top proposition, takes the dashed/low/false or solid/high/true branch. The procedure is called recursively on the resulting node until a terminal is reached. Because in a BDD each variable occurs only once on a path, it can be used to compute the probability of the DNF formula with Algorithm 1.

Example 3. Using Algorithm 1 and Figure 1a, it is easy to see that for the strategy of bringing an umbrella and not wearing a raincoat, the success probability of staying dry is $\text{P}(dry|\sigma(\mathcal{DT})) = 0.7 + 0.3 \cdot 0.5 = 0.85$ and that $\text{Util}(dry|\sigma, \mathcal{DT}) = 60 \cdot 0.85 = 51$. Using the BDD for `broken_umbrella`, we can calculate $\text{Util}(\sigma, \mathcal{DT}) = 51 + (0.15 \cdot (-40)) + (-2) = 43$.



(a) `dry` (b) `broken_umbrella`

Figure 1: BDDs for `dry` and `broken_umbrella`.

4 Solving Decision Problems

When faced with a decision problem, one is interested in computing the optimal strategy, that is, according to the maximum expected utility principle, finding σ^*:

$$\sigma^* = \text{argmax}_\sigma(\text{Util}(\sigma, \mathcal{DT})).$$

This strategy is the *solution* to the decision problem. We will first introduce an exact algorithm for finding deterministic solutions and then outline two ways to approximate the optimal strategy.

Exact Algorithm

Our exact algorithm makes use of Algebraic Decision Diagrams (ADD) (Bahar et al. 1997) to efficiently represent the

Algorithm 2 Finding the exact solution for $\mathcal{DT}$

function EXACTSOLUTION(Theory $\mathcal{DT}$)
  $\text{ADD}^{\text{util}}_{tot}(\sigma) \leftarrow$ a 0-terminal
  for each $(u \rightarrow r) \in \mathcal{U}$ **do**
    $\text{BDD}_u(\mathcal{DT}) \leftarrow \text{BINARYDD}(u)$
    $\text{ADD}_u(\sigma) \leftarrow \text{PROBABILITYDD}(\text{BDD}_u(\mathcal{DT}))$
    $\text{ADD}^{\text{util}}_u(\sigma) \leftarrow r \cdot \text{ADD}_u(\sigma)$
    $\text{ADD}^{\text{util}}_{tot}(\sigma) \leftarrow \text{ADD}^{\text{util}}_{tot}(\sigma) \oplus \text{ADD}^{\text{util}}_u(\sigma)$
  let t_{max} be the terminal node of $\text{ADD}^{\text{u}}_{tot}(\sigma)$ with the highest utility
  let p be a path from t_{max} to the root of $\text{ADD}^{\text{u}}_{tot}(\sigma)$
  return the boolean decisions made on p

function PROBABILITYDD(BDD-node n)
  if n is the 1-terminal **then return** a 1-terminal
  if n is the 0-terminal **then return** a 0-terminal
  let h and l be the high and low children of n
  $\text{ADD}_h \leftarrow \text{PROBABILITYDD}(h)$
  $\text{ADD}_l \leftarrow \text{PROBABILITYDD}(l)$
  if n represents a decision d **then**
    return $\text{ITE}(d, \text{ADD}_h, \text{ADD}_l)$
  if n represents a fact with probability p **then**
    return $(p_n \cdot \text{ADD}_h) \oplus ((1 - p_n) \cdot \text{ADD}_l)$

utility function $Util(\sigma, \mathcal{DT})$. ADDs generalize BDDs such that leaves may take on any value and can be used to represent any function from booleans to the reals $[0, 1]^n \rightarrow \mathbb{R}$. Operations on ADDs relevant for this paper are the scalar multiplication $c \cdot g$ of an ADD g with the constant c, the addition $f \oplus g$ of two ADDs, and the *if-then-else* test $\text{ITE}(b, f, g)$.

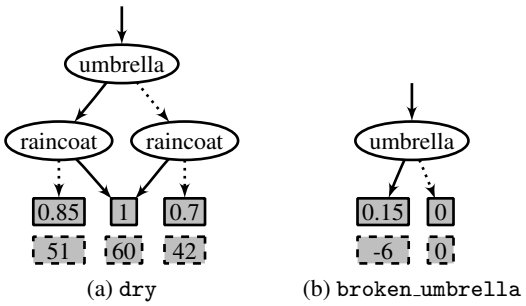
Using these primitive operations we construct:

1. $\text{BDD}_u(\mathcal{DT})$ representing $\mathcal{DT} \models u$ as a function of the probabilistic and decision facts in $\mathcal{DT}$.
2. $\text{ADD}_u(\sigma)$ representing $\text{P}(u|\sigma, \mathcal{DT})$ as function of σ.
3. $\text{ADD}^{\text{util}}_u(\sigma)$ representing $\text{Util}(u|\sigma, \mathcal{DT})$ as function of σ
4. $\text{ADD}^{\text{util}}_{tot}(\sigma)$ representing $\text{Util}(\sigma, \mathcal{DT})$ as function of σ

These four diagrams map to the steps in the for-loop of Algorithm 2. The first step builds the BDD for the query u_i as described in Section 3. The difference is that the nodes representing decisions get marked as such, instead of getting a 0/1-probability assigned. In the second step, this BDD is transformed into an ADD using Algorithm 2, an adaptation of Algorithm 1. The resulting ADD contains as internal nodes only decision nodes and the probabilities are propagated into the leafs. The third step scales $\text{ADD}_u(\sigma)$ by the reward for u as in Equation 2.

Example 4. Figure 2 shows the $\text{ADD}_{dry}(\sigma)$ and $\text{ADD}_{broken\_umbrella}(\sigma)$, constructed using Algorithm 2 from the BDDs in Figure 1. The former confirms that $\text{P}(dry|\sigma(\mathcal{DT})) = 0.85$ for the strategy given in Example 3. The transformation to $\text{ADD}^{\text{util}}_u(\sigma)$ is done by replacing the terminals by their dashed alternatives.

Finally, in the fourth step, this ADD is added to the global sum $\text{ADD}^{\text{util}}_{tot}(\sigma)$ according to Equation 3, modeling the ex-



(a) `dry` (b) `broken_umbrella`

Figure 2: $\mathrm{ADD}_{dry}(\sigma)$ and $\mathrm{ADD}_{broken\_umbrella}(\sigma)$. The alternative, dashed terminals belong to $\mathrm{ADD}_{u}^{\mathrm{util}}(\sigma)$.

pected utility of the entire DTPROBLOG theory. From the final ADD, the globally optimal strategy σ^* is extracted by following a path from the leaf with the highest value to the root of the ADD. Because ADDs provide a compressed representation and efficient operations, the exact solution algorithm is able to solve more problems in an exact manner than could be done by naively enumerating all possible strategies.

Example 5. Figure 3 shows $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$. It confirms that the expected utility of the strategy from Example 3 is 43. It turns out that this is the optimal strategy. For wearing a raincoat, the increased probability of staying dry does not outweigh the cost. For bringing an umbrella, it does.

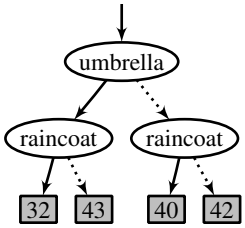


Figure 3: $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$ for $\mathrm{Util}(\sigma, \mathcal{DT})$

Sound Pruning

Algorithm 2 can be further improved by avoiding unnecessary computations. The algorithm not only finds the best strategy, but also represents the utility of all possible strategies. Since we are not interested in those values, they can be removed from the ADDs when they become irrelevant for finding the optimal value. The idea is to keep track of the maximal utility that can be achieved by the utility attributes not yet added to the ADD. While adding further ADDs, all nodes of the intermediate ADD that can not yield a value higher than the current maximum can be pruned. For this, we define the maximal impact of a utility attribute to be

$$\mathrm{Im}(u_i) = \max(\mathrm{ADD}_{u_i}^{\mathrm{util}}(\sigma)) - \min(\mathrm{ADD}_{u_i}^{\mathrm{util}}(\sigma)),$$

where $\max$ and $\min$ are the maximal and minimal terminals. Before adding $\mathrm{ADD}_{u_i}^{\mathrm{util}}(\sigma)$ to the intermediate $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$, we merge all terminals from $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$ with a value below

$$\max(\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)) - \sum_{j \geq i} \mathrm{Im}(u_j)$$

by setting their value to minus infinity. These values are so low that even in the best case, they will never yield the maximal value in the final ADD. By sorting the utility attributes by decreasing values of $\mathrm{Im}(u)$, even more nodes are removed from the ADD. This improvement still guarantees that an optimal solution is found. In the following sections, we will show two improvements which will not have this guarantee, but are much faster. The two improvements can be used together or independently.

Local Search

Solving a DTPROBLOG program is essentially a function optimization problem for $\mathrm{Util}(\sigma, \mathcal{DT})$ and can be formalized as a search problem in the strategy space. We apply a standard greedy hill climbing algorithm that searches for a locally optimal strategy. This way, we avoid the construction of the ADDs in steps 2-4 of Algorithm 2. The search starts with a random strategy and iterates repeatedly over the decisions. It tries to flip a decision, forming σ'. If $\mathrm{Util}(\sigma', \mathcal{DT})$ improves on the previous utility, σ' is kept as the current best strategy. The utility value can be computed using Equations (3) and (2). To efficiently calculate $\mathrm{Util}(\sigma', \mathcal{DT})$, we use the BDDs generated by the BINARYDD function of Algorithm 2. During the search, the BDDs can be kept fixed. Only the probability values for those BDDs that are effected by the changed decision have to be updated.

Approximative Utility Evaluation

The second optimization is concerned with the first step of Algorithm 2 that finds all proofs for the utility attributes. In large decision problems this quickly becomes intractable. A number of approximative inference methods exist for ProbLog (Kimmig et al. 2008), among which is the k-best approximation. The idea behind it is that, while there are many proofs, only a few contribute significantly to the total probability. It incorporates only those k proofs where the product of the random variables that make up the proof is highest, computing a lower bound on the success probability of the query. The required proofs are found using a *branch-and-bound* algorithm. Similarly, we use the k-best proofs for the utility attributes to build the BDDs and ADDs in the strategy solution algorithms. This reduces the runtime and the complexity of the diagrams. For sufficiently high values of k, the solution strategy found will be optimal.

5 Experiments

The experiments were set up to answer the questions: **(Q1)** Does the exact algorithm perform better than naively calculating the utility for all possible strategies? **(Q2)** How does local search compare to the exact algorithm in terms of runtime and solution quality? **(Q3)** What is the trade off between runtime and solution quality for different values of k, using the k-best proofs approximation? **(Q4)** Do the algorithms scale to large, real world problems?

To answer these questions we tested the algorithms on the viral marketing problem, a prime example of relational non-sequential decision making. The viral marketing problem was formulated by Domingos and Richardson (2001) and

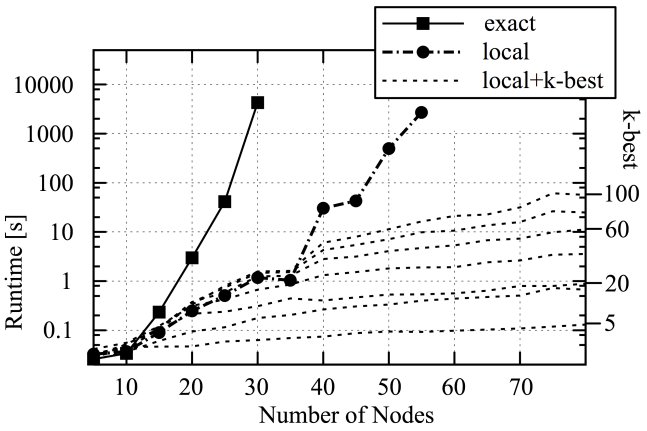


Figure 4: Runtime of solving viral marketing in random graphs of increasing size. The values are averaged over three runs on four different graphs of the same size. Methods differ in the search algorithm and the number of proofs used.

used in experiments with Markov Logic (Nath and Domingos 2009). Given a social network structure consisting of $\texttt{trusts}(\texttt{a},\texttt{b})$ relations, the decisions are whether or not to market to individuals in the network. A reward is given for people buying the product and marketing to an individual has a cost. People that are marketed or that trust someone that bought the product may buy the product. In DTProbLog, this problem can be modeled as:

```
? :: market(P) :- person(P).
0.4 :: viral(P,Q).
0.3 :: from_marketing(P).
market(P) → −2 :- person(P).
buys(P) → 5 :- person(P).
buys(P) :- market(P), from_marketing(P).
buys(P) :- trusts(P,Q), buys(Q), viral(P,Q).
```

The example shows the use of syntactic sugar in the form of templates for decisions and utility attributes. It is allowed to make decision facts and utility attributes conditional on a body of literals. For every substitution for which the body succeeds, a corresponding decision fact or utility attribute is constructed. We impose the restriction that these bodies can only depend on deterministic facts. For instance, in the example, there is one `market` decision for each person.

We tested the algorithms[2] on the viral marketing problem using a set of synthetic power law random graphs, known to resemble social networks (Barabasi and Bonabeau 2003). The average number of edges or trust relations per person was chosen to be 2. Figure 4 shows the runtime for different solution algorithms on graphs with increasing node count. For reproducibility, we start the local search algorithm from a zero-vector for σ and flip decision in a fixed order.

[2]Implemented in YAP 6 `http://www.dcc.fc.up.pt/~vsc/Yap/` for the Prolog part and simpleCUDD 2.0.0 `http://www.cs.kuleuven.be/~theo/tools/simplecudd.html` for the decision diagrams.

This allows us to answer the first three questions: **(Q1)** While the exact algorithm is fast for small problems and guaranteed to find the optimal strategy, it becomes unfeasible on networks with more than 30 nodes. The 10-node problem is the final one solvable by the naive approach and takes over an hour to compute. Solving the 30-node graph in a naive manner would require over a billion inference steps, which is intractable. The exact algorithm clearly outperforms a naive approach. **(Q2)** Local search solves up to 55-node problems when it takes all proofs into account and was able to find the globally optimal solution for those problems where the exact algorithm found a solution. This is not necessarily the case for other decision problems with more deterministic dependencies. **(Q3)** After the 55-node point, the BDDs had to be approximated by the k-best proofs. For higher values of k, search becomes slower but is more likely to find a better strategy. For k larger than 20 the utility was within 2% of the best found policy for all problem sizes. To answer **(Q4)**, we experimented on a real world dataset of trust relations extracted from the Epinions[3] social network website (Richardson and Domingos 2002). The network contains 75888 people that each trust 6 other people on average. Local search using the 17-best proofs finds a locally optimal strategy for this problem in 16 hours.

6 Related Work

Several AI-subfields are related to DTProbLog, either because they focus on the same problem setting or because they use compact data-structures for decision problems. Closely related is the *independent choice logic* (ICL) (Poole 1997), which shares its *distribution semantics* (Sato 1995) with ProbLog, and which can represent the same kind of decision problems as DTProbLog. Similar to DTProbLog being an extension of an existing language ProbLog, so have two related system been extended towards utilities recently. Nath and Domingos (2009) introduce *Markov logic decision networks* (MLDN) based on Markov logic networks and Chen and Muggleton (2009) extend *stochastic logic programs* (SLP) towards *decision-theoretic logic programs* (DTLP). The DTLP approach is close to the syntax and semantics of DTProbLog, although some restrictions are put on the use of decisions in probabilistic clauses. Chen and Muggleton also devise a parameter learning algorithm derived from SLPs. Nath and Domingos (2009) introduce *Markov logic decision networks* (MLDN) based on Markov logic networks. Many differences between MLNs and ProbLog exist and these are carried over to DTProbLog. Yet we are able to test on the same problems, as described earlier. Whereas DTProbLog's inference and search can be done both exact and approximative, MLDN's methods only compute approximate solutions. Some other formalisms too can model decision problems e.g. IBAL (Pfeffer 2001), and *relational decision networks* (Hsu and Joehanes 2004)). Unlike DTProbLog, DTLPs, ICL and *relational decision networks* currently provide no implementation based on efficient data structures tailored towards decision problems and hence, no results are reported on a large

[3]`http://www.epinions.com/`

problem such as the Epinions dataset. IBAL has difficulties to represent situations in which properties of different objects are mutually dependent, like in the viral marketing example.

DTPROBLOG is also related to various works on Markov decision processes. In contrast to DTProbLog, these are concerned with sequential decision problems. Nevertheless, they are related in the kind of techniques they employ. For instance, for *factored Markov decision processes* (FMDPs), SPUDD (Hoey et al. 1999) also uses ADDs to represent utility functions, though it cannot represent *relational* decision problems and is not a programming language. On the other hand, there exist also *first-order* (or, *relational*) *Markov decision processes* (FOMDP), see (van Otterlo 2009). Techniques for FOMDPs have often been developed by upgrading corresponding algorithms for FMDPs to the relational case, including the development of compact *first-order decision diagrams* by Wang, Joshi, and Khardon (2008) and Sanner and Boutilier (2009). While first-order decision diagrams are very attractive, they are not yet as well understood and well developed as their propositional counterparts. A unique feature of DTPROBLOG (and the underlying ProbLog system) is that it solves *relational* decision problems by making use of efficient *propositional* techniques and data structures.

Perhaps the most important question for further research is whether and how DTPROBLOG and its inference algorithms can be adapted for use in sequential decision problems and FOMDPs. DTPROBLOG is, in principle, expressive enough to model such problems because it is a programming language, allowing the use of structured terms such as lists or natural numbers to represent sequences and time. However, while DTPROBLOG can essentially represent such problems, a computational investigation of the effectiveness of its algorithms for this type of problem still needs to be performed and may actually motivate further modifications or extensions of DTPROBLOG's engine, such as tabling (currently under development for ProbLog) or the use of first order decision diagrams.

7 Conclusions

A new decision-theoretic probabilistic logic programming language, called DTPROBLOG, has been introduced. It is a simple but elegant extension of the probabilistic Prolog ProbLog. Several algorithms for performing inference and computing the optimal strategy for a DTPROBLOG program have been introduced. This includes an exact algorithm to compute the optimal strategy using binary and algebraic decisions diagrams as well as two approximation algorithms. The resulting algorithms have been evaluated in experiments and shown to work on a real life application.

Acknowledgments This work was supported in part by the Research Foundation-Flanders (FWO-Vlaanderen) and the GOA project 2008/08 Probabilistic Logic Learning.

References

Bahar, R.; Frohm, E.; Gaona, C.; Hachtel, G.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebraic Decision Diagrams and Their Applications. *Formal Methods in System Design* 10:171–206.

Barabasi, A., and Bonabeau, E. 2003. Scale-free networks. *Scientific American* 288(5):50–59.

Bryant, R. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on computers* 35(8):677–691.

Chen, J., and Muggleton, S. 2009. Decision-Theoretic Logic Programs. In *Proceedings of ILP*.

De Raedt, L.; Frasconi, P.; Kersting, K.; and Muggleton, S., eds. 2008. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer.

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

Domingos, P., and Richardson, M. 2001. Mining the network value of customers. In *Proceedings of KDD*, 57–66.

Flach, P. 1994. *Simply Logical: Intelligent Reasoning by Example*.

Getoor, L., and Taskar, B. 2007. *Introduction to statistical relational learning*. MIT Press.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. *Proceedings of UAI* 279–288.

Hsu, W., and Joehanes, R. 2004. Relational Decision Networks. In *Proceedings of the ICML Workshop on Statistical Relational Learning*.

Kimmig, A.; Santos Costa, V.; Rocha, R.; Demoen, B.; and De Raedt, L. 2008. On the efficient execution of ProbLog programs. In *Proceedings of ICLP*.

Nath, A., and Domingos, P. 2009. A Language for Relational Decision Theory. In *Proceedings of SRL*.

Pfeffer, A. 2001. IBAL: A probabilistic rational programming language. In *Proceedings of IJCAI*, volume 17, 733–740.

Poole, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 94(1-2):7–56.

Richardson, M., and Domingos, P. 2002. Mining knowledge-sharing sites for viral marketing. In *Proceedings of KDD*, 61.

Russell, S., and Norvig, P. 2003. *Artificial intelligence: A modern approach.* Prentice Hall.

Sanner, S., and Boutilier, C. 2009. Practical solution techniques for first-order MDPs. *Artificial Intelligence* 173(5-6):748–788.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *Proceedings of ICLP*, 715–729.

van Otterlo, M. 2009. *The logic of adaptive behavior*. IOS Press, Amsterdam.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research* 31(1):431–472.