Variable Compression in ProbLog

Theofrastos Mantadelis and Gerda Janssens

Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract. In order to compute the probability of a query, ProbLog represents the proofs of the query as disjunctions of conjunctions, for which a Reduced Ordered Binary Decision Diagram (ROBDD) is computed. The paper identifies patterns of Boolean variables that occur in Boolean formulae, namely AND-clusters and OR-clusters. Our method compresses the variables in these clusters and thus reduces the size of ROBDDs without affecting the probability.
We give a polynomial algorithm that detects AND-clusters in disjunctive normal form (DNF) Boolean formulae, or OR-clusters in conjunctive normal form (CNF) Boolean formulae.
We do an experimental evaluation of the effects of AND-cluster compression for a real application of ProbLog. With our prototype implementation we have a significant improvement in performance (up to 87%) for the generation of ROBDDs. Moreover, compressing AND-clusters of Boolean variables in the DNFs makes it feasible to deal with ProbLog queries that give rise to larger DNFs.
Keywords: ProbLog, Statistical Relation Learning, Probabilistic Logic Programming, Variable Compression, Binary Decision Diagrams

1 Introduction

ProbLog [1, 2] is a probabilistic framework that extends Prolog with probabilistic facts. ProbLog computes the probability of a query in two main steps. First, ProbLog collects the probabilistic facts for each SLD proof of the query. Each probabilistic fact is represented by a Boolean variable, each proof by the conjunction of probabilistic facts used in the proof, and the set of all proofs by a disjunction of conjunctions (a DNF). In the second step, ProbLog uses ROBDDs [3, 4] to calculate the success probability of the query. Note that assessing the probability of a DNF Boolean formula is a #P-complete problem [5] and using ROBBDs is a state-of-the-art approach [6].

For typical ProbLog applications, generating a ROBDD can become one of the limiting factors. The size of the constructed ROBDD depends heavily on the variable ordering. There has been a lot of research on finding efficient variable orderings by using static [7, 8] and dynamic heuristics [9, 10]. In this paper we present variable compression as a complementary approach to construct smaller ROBDDs. We observed patterns (AND-clusters, OR-clusters) in the ROBDDs that make it possible to replace a set of Boolean variables with a single new one

without affecting the final probability. To benefit from the variable compression, the clusters should be discovered before the actual ROBDD generation.

The paper has two main contributions. The first contribution is the definition of the AND-clusters and OR-clusters, their usage in assessing the probability of a DNF Boolean formula, and their usage for compressing ROBDDs. The second contribution is the Book Marking algorithm that detects AND-clusters in ProbLog set of proofs.

We also evaluate experimentally the effects of the AND-clusters in a typical ProbLog application [1, 11]. Our experiments show the impact of the AND-cluster compression: the number of variables in the ROBDD is on average reduced by 28% and the time performance of the generation of the ROBDDs improves on average by 41%. The AND-cluster compression also allowed us to compute queries that caused timeouts. In our benchmarks AND-cluster compression is beneficial for larger DNFs where the cost of executing the bookmarking algorithm is lower than the time gain we have during ROBDD generation.

We briefly introduce ProbLog in Section 2 and explain how ROBDDs are used. In Section 3 we define AND-clusters, OR-clusters and present their use in a probabilistic context. The Book Marking algorithm for AND-clusters is in Section 4. The experiments follow in Section 5 and the complexity analysis is in Section 6. Finally, Section 7 concludes.

2 ProbLog and its use of ROBDDs

2.1 The ProbLog Language

A ProbLog program T [2] consists of a set of labelled ground facts $p_i :: pf_i$ together with a set of definite clauses. Each such fact pf_i is true with probability p_i, that is, these facts correspond to random variables, which are assumed to be mutually independent. Together, they thus define a distribution over subsets of $L_T = \{pf_1, \ldots, pf_n\}$. The definite clauses add arbitrary *background knowledge* (BK) to those sets of *logical* facts. Given the one-to-one mapping between ground definite clause programs and Herbrand interpretations, a ProbLog program also defines a distribution over its Herbrand interpretations.

ProbLog inference calculates the *success probability* $P_s(q|T)$ of a query q in a ProbLog program T, that is, the probability that the query q is *provable* in a logic program that combines BK with a randomly sampled subset of L_T.

Figure 1 shows a small ProbLog program encoding a probabilistic graph and the graph that is represented by the probabilistic facts `edge/2`. The success probability of `path(1,3)` corresponds to the probability that a randomly sampled subgraph contains at least one of the four possible paths from node `1` to node `3`.

2.2 Program Execution in ProbLog

ProbLog programs are executed in two steps. Given a ProbLog program T and a query q, the first step, *SLD-resolution*, collects all proofs for query q in $BK \cup L_T$.

```
0.5::edge(1, 2). % x0   0.4::edge(1, 4). % x1   0.7::edge(2, 3). % x2
0.8::edge(2, 6). % x3   0.9::edge(4, 5). % x4   0.7::edge(5, 2). % x5
0.6::edge(5, 7). % x6   0.4::edge(6, 3). % x7   0.3::edge(6, 7). % x8
path(X, Y):- path(X, Y, [X]).
path(X, Y, _):- edge(X, Y).
path(X, Y, A):- edge(X, Z), \+ member(Z, A), path(Z, Y, [Z|A]).
% Query: problog_exact(path(1, 3), Probability)
```

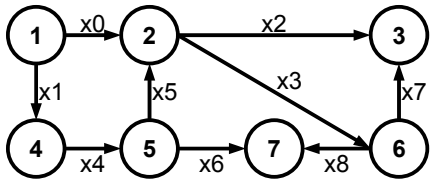


Fig. 1. Example ProbLog program path/2 and its graph.

Proofs are stored as lists of probabilistic facts in a *trie data structure*. This trie represents the proofs of the query q in a compact way as it exploits prefix sharing between proofs. The usage of tries is not important for this paper.

In our example, SLD resolution finds four proofs for the query `path(1,3)` which are represented by the following lists of probabilistic `edge/2` facts:

```
edge(1,2),edge(2,3)
edge(1,2),edge(2,6),edge(6,3)
edge(1,4),edge(4,5),edge(5,2),edge(2,3)
edge(1,4),edge(4,5),edge(5,2),edge(2,6),edge(6,3)
```

In general these lists of probabilistic facts express the Boolean formula:

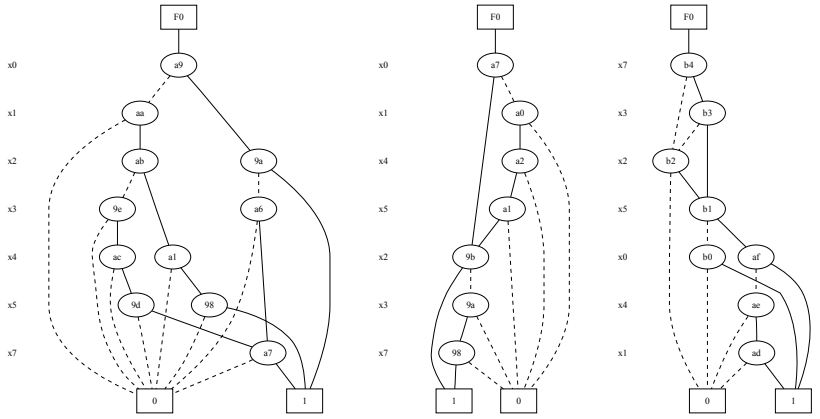
$$\bigvee_{pr_j \in proofs} \left(\bigwedge_{pf_i \in pr_j} pf_i \right) \tag{1}$$

where the pf_i represent the probabilistic facts used in proof pr_j. Using the x_i to represent the `edge/2` facts as indicated in the ProbLog program, the DNF for the `path(1,3)` query is the formula $(x_0 \wedge x_2) \vee (x_0 \wedge x_3 \wedge x_7) \vee (x_1 \wedge x_4 \wedge x_5 \wedge x_2) \vee (x_1 \wedge x_4 \wedge x_5 \wedge x_3 \wedge x_7)$. In order to compute the correct probability for (1), ProbLog faces the disjoint sum problem [6]. ProbLog solves this in its second step, namely by the transformation of the disjunction of conjunctions into mutually disjoint conjunctions by constructing a ROBDD for (1).

A ROBDD for the `path(1,3)` example of Figure 1 is given in Figure 2a. The topmost circular node in the ROBDD corresponds to the probabilistic fact x_0 and is called a variable node. A variable node has two successors pointed to by the **high** edge and the **low** edge. Edges are implicitly directed: they point downwards. The ROBDD that is rooted at the low successor represents the Boolean expression that is yielded by substituting "false" for the variable. The high successor represents the Boolean expression that is yielded by substituting

"true". The "true" node 1 and the "false" node 0 represent whether the binary formula is satisfied or not. The paths from the root to node 1 give the disjoint sum as a disjunction of disjoint conjunctions.

The computation of the probability is bottom up and linear in the size of the ROBDD. Details are in [2]. ProbLog computes that 0.498296 is the exact probability that a path from node 1 to node 3 exists.



(a) Bad ordering. x0-x1-x2-x3-x4-x5-x7

(b) Good ordering. x0-x1-x4-x5-x2-x3-x7

(c) ProbLog ordering. x7-x3-x2-x5-x0-x4-x1

Fig. 2. ROBDD for the query `path(1,3)`.

Note that in general a variable can have multiple nodes in a ROBDD. For example, in Figure 2a the variables x2, x3, x4, x5 have two nodes each. A ROBDD imposes an order on the Boolean variables and the different variables appear in that order in all the paths of the ROBDD. ROBDDs are reduced which means that two nodes never have the same successors if they are nodes of the same variable and that no node has the same high and low successor. These reductions do not perform the variable compression we are aiming at.

3 Variable Compression

3.1 Motivation

ProbLog uses ROBDDs to compute the success probability of a query. While the complexity of the calculation of the probability is linear in terms of the size of the ROBDD, the generation of the ROBDD is NP-hard.

It is well-known that the variable ordering used to construct the ROBDD for a Boolean formula has an impact on the size of the ROBDD. For our `path(1,3)` example, Figure 2a until 2c use different variable orderings. The orderings that give rise to smaller ROBDDs are called good orderings: constructing smaller

ROBDDs takes less time and space and also the computation of the probability is faster. State-of-the-art ROBDD tools use heuristics to decide about the variable ordering, whose search space is exponential.

We reduce the search space for the variable ordering by decreasing the number of variables in the Boolean formula, namely by replacing subsets of variables by new representative variables. We call this **variable compression**. We can only do variable compression if we do not affect the probability.

We discovered sets of variables in the ROBDDs for which we can compute the contribution of such a set of variables to the probability of the ROBDD independently from the rest of the ROBBD. For example, the set of variables x1, x4 and x5 in Figure 2b. This implies that we can replace those three variables by a new representative variable, whose probability is computed from the probabilities of x1, x4 and x5. Later in this section we define this set as an AND-cluster.

In order to do variable compression before ROBDD generation, we need to detect these patterns in the Boolean formulae, or in the case of ProbLog at the level of the DNF (1). It turns out that in the proofs of `path(1,3)` either the probabilistic facts corresponding to x1, x4 and x5 appear all three together in a proof, or none of them occurs. The AND-clusters are determined for particular DNFs and as such they are query-dependent. For path(1,7), x1, x4 and x5 no longer form an AND-cluster as we also have a proof `edge(1,4),edge(4,5),edge(5,7)` that does not contain x5. Now only x1 and x4 form an AND-cluster.

As the AND-clusters are query-dependent, they do not appear in the ProbLog source program itself. Although one could be tempted to replace the facts `edge(1,4),edge(4,5)` by a single one, this is not a good idea because path/2 queries could have 4 as a starting node.

3.2 Cluster Definitions

We define two kinds of clusters and proof that their compression does not effect the final probability. We define the clusters in terms of the ROBDDs because the patterns can also be valuable for other application areas that use ROBDDs.

Definition 1. *Let F be a Boolean formula with variables $v_1, \ldots, v_l$. The variables $\{x_1, \ldots, x_k\} \subseteq \{v_1, \ldots, v_l\}, k > 1$, form an **AND-cluster** if there exists a variable ordering such that the ROBDD R of F*

1. *has only one node n_i for variable $x_i, 1 \leq i \leq k$,*
2. *node n_j has as only incoming edge the high edge of node $n_{j-1}, 2 \leq j \leq k$,*
3. *and the low edges of the nodes $\{n_1, ..., n_k\}$ connect to the same node in R.*

Definition 2. *Let F be a Boolean formula with variables $v_1, \ldots, v_l$. The variables $\{x_1, \ldots, x_k\} \subseteq \{v_1, \ldots, v_l\}, k > 1$, form an **OR-cluster** if there exists a variable ordering such that the ROBDD R of F*

1. *has only one node n_i for variable $x_i, 1 \leq i \leq k$,*
2. *node n_j has as only incoming edge the low edge of node $n_{j-1}, 2 \leq j \leq k$,*
3. *and the high edges of the nodes $\{n_1, ..., n_k\}$ connect to the same node in R.*

In a probabilistic framework like ProbLog that uses ROBDDs to calculate probabilities, each ROBDD variable has an assigned probability. To be able to compress the clusters of variables to new representative variables, we need to compute the probabilities of the representative variables such that the probabilty we compute for the ROBDD as a whole does not change.

Theorem 1 (Probability of AND-cluster). *To replace an AND-cluster* $\{x_1, \ldots, x_n\}$ *by a representative variable* V *with probability* $P_V = P_{AND}(\{x_1, \ldots, x_n\}) = \prod_{i=1}^{n} P(x_i)$ *does not change the probability of the ROBBD as a whole.*

Proof. First consider the simple case of a ROBDD that consist of exactly one AND-cluster, $\{x_1, \ldots, x_n\}$. The probability of this ROBDD is $P(x_1) \cdot ... \cdot P(x_i) \cdot ... \cdot P(x_n) \cdot 1 + (1 - P(x_1) \cdot ... \cdot P(x_i) \cdot ... \cdot P(x_n)) \cdot 0 = \prod_{i=1}^{n} P(x_i)$. But in general, an AND-cluster has an outgoing high edge to a part T with P_T and its low edges connect to a part F with P_F. The probability of the ROBDD part that includes the AND-cluster can be generalised as $P = P(x_1) \cdot ... \cdot P(x_i) \cdot ... \cdot P(x_n) \cdot P_T + (1 - P(x_1) \cdot ... \cdot P(x_i) \cdot ... \cdot P(x_n)) \cdot P_F = P_T \cdot \prod_{i=1}^{n} P(x_i) + P_F - P_F \cdot \prod_{i=1}^{n} P(x_i) = (P_T - P_F) \cdot \prod_{i=1}^{n} P(x_i) + P_F$. If we replace the AND-cluster with a new representative variable V with P_V and calculate the probability, we get $P = P_V \cdot P_T + (1 - P_V) \cdot P_F = P_V \cdot P_T + P_F - P_V \cdot P_F = (P_T - P_F) \cdot P_V + P_F$. Therefore $P_V = P_{AND}(\{x_1, \ldots, x_n\}) = \prod_{i=1}^{n} P(x_i)$.

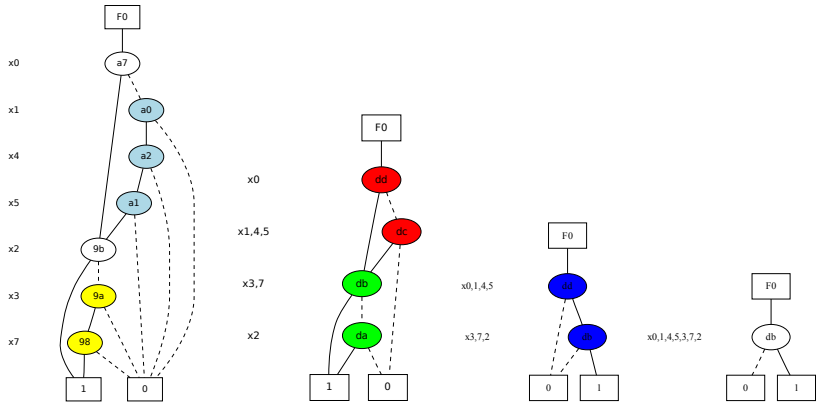
Theorem 2 (Probability of an OR-cluster). *To replace an OR-cluster* $\{x_1, \ldots, x_n\}$ *to the representative variable* V *with probability* $P_V = P_{OR}(\{x_1, \ldots, x_n\}) = P(x_1) + (1 - P(x_1)) \cdot P_{OR}(\{x_2, \ldots, x_n\})$ *and* $P_{OR}(\{x_n\}) = P(x_n)$ *does not change the probability of the ROBBD as a whole.*

Proof. First consider the simple case of a ROBDD that consist of exactly one OR-cluster, $\{x_1, \ldots, x_n\}$. The probability of this ROBDD is $P(x_1) \cdot 1 + (1 - P(x_1)) \cdot (P(x_2) \cdot 1 + (1 - P(x_2)) \cdot \ldots \cdot (P(x_i) \cdot 1 + (1 - P(x_i)) \cdot \ldots \cdot (P(x_n) \cdot 1 + (1 - P(x_n)) \cdot 0)) \ldots) = P(x_1) + (1 - P(x_1)) \cdot P_{OR}(\{x_2, \ldots, x_n\})$. But in general an OR-cluster has its high edges to a part T with P_T and an outgoing low edge to a part F with P_F. The probability can be generalised as $P = P(x_1) \cdot P_T + (1 - P(x_1)) \cdot (P(x_2) \cdot P_T + (1 - P(x_2)) \cdot \ldots \cdot (P(x_i) \cdot P_T + (1 - P(x_i)) \cdot \ldots \cdot (P(x_n) \cdot P_T + (1 - P(x_n)) \cdot P_F)) \ldots) = (P(x_1) + (1 - P(x_1)) \cdot P(x_2) + \ldots + (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_{i-1})) \cdot P(x_i) + \ldots + (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_{n-1})) \cdot P(x_n)) \cdot P_T + (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_n)) \cdot (P_F / P_T)$. If we replace the OR-cluster with a new representative variable V with P_V and calculate the probability, we get $P = P_V \cdot P_T + (1 - P_V) \cdot P_F$ if we replace $P_V = P(x_1) + (1 - P(x_1)) \cdot P(x_2) + \ldots + (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_{i-1})) \cdot P(x_i) + \ldots + (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_{n-1})) \cdot P(x_n)$ then we need to prove that $1 - P_V = (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_n)) \Rightarrow 1 - (P(x_1) + (1 - P(x_1)) \cdot P(x_2) + \ldots + (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_{i-1})) \cdot P(x_i) + \ldots + (1 -$

$P(x_1)) \cdot \ldots \cdot (1 - P(x_{n-1})) \cdot P(x_n)) = (1 - P(x_1)) \cdot \ldots \cdot (1 - P(x_n))$. Finally by using the distribution rule we see that the previous formula is a tautology. Therefore $P_V = P_{OR}(\{x_1, \ldots, x_n\}) = P(x_1) + (1 - P(x_1)) \cdot P_{OR}(\{x_2, \ldots, x_n\})$.

3.3 Using the Clusters for Variable Compression

We illustrate the variable compression with our `path(1,3)` example. In Figure 3a we have two AND-clusters, $\{x1, x4, x5\}$ and $\{x3, x7\}$. After compression we obtain the ROBDD in Figure 3b with two new Boolean variables $x1, 4, 5$, $x3, 7$ and their associated probabilities $P(x1, 4, 5)$[1], $P(x3, 7)$[2]. After AND-compression we now have two OR-clusters, $\{x0, x1, 4, 5\}$ and $\{x3, 7, x2\}$ as shown in Figure 3b; by further compressing them we get the ROBDD in Figure 3c with two new Boolean variables $x0, 1, 4, 5$, $x3, 7, 2$ and their probabilities $P(x0, 1, 4, 5)$[3], $P(x3, 7, 2)$[4]. Finally, by compressing the single AND-cluster $\{x0, 1, 4, 5, x3, 7, 2\}$ of the ROBDD in Figure 3c we end up with the ROBDD in Figure 3d that has a single Boolean variable $x0, 1, 4, 5, 3, 7, 2$ and probability $P(x0, 1, 4, 5, 3, 7, 2)$[5].



(a) AND-clusters. (b) OR-clusters. (c) AND-cluster. (d) Compressed.

Fig. 3. Compressing ROBDD of `path(1,3)`. Notation: coloured nodes represent clusters.

Not all ROBDDs can be compressed to a single variable, iterating AND/OR-cluster based variable compression can lead to an easier to construct ROBDD. We want to use variable compression to be able to deal with queries that caused timeouts. So, we are willing to pay a certain cost to detect the clusters. In order

[1] $P(x1, 4, 5) = P_{AND}(\{x1, x4, x5\}) = 0.4 \cdot 0.9 \cdot 0.7 = 0.252$
[2] $P(x3, 7) = P_{AND}(\{x3, x7\}) = 0.8 \cdot 0.4 = 0.32$
[3] $P(x0, 1, 4, 5) = P_{OR}(\{x0, x1, 4, 5\}) = 0.5 + (1 - 0.5) \cdot 0.252 = 0.626$
[4] $P(x3, 7, 2) = P_{OR}(\{x3, 7, x2\}) = 0.32 + (1 - 0.32) \cdot 0.7 = 0.796$
[5] $P(x0, 1, 4, 5, 3, 7, 2) = P_{AND}(\{x0, 1, 4, 5, x3, 7, 2\}) = 0.626 \cdot 0.796 = 0.498296$

to use our variable compression in practise, we first have to detect AND-clusters in DNFs. In the rest of this paper we focus on AND-clusters and we already obtain promising results with AND-cluster variable compression. The detection of OR-clusters in the DNFs is part of future work.

4 Discovering AND-clusters

To be able to benefit from AND-cluster compression, we need to identify them before the ROBDD generation. Fortunately, AND-clusters also appear in the DNF representing the proofs: either all the probabilistic facts of an AND-cluster appear in a proof, or none of them. A naive approach to detect AND-clusters is to find longest common subsequences (LCS) in conjunctions of the DNF, however this is an NP-hard problem [12]. As our problem is a special case of the LCS we can do better.

Lemma 1. *Every set of probabilistic facts* $\{pf_1, ..., pf_n\}$ *in a set of proofs* $\{pr_1, \ldots, pr_m\}$ *satisfying* $\forall pf_i \in \{pf_1, ..., pf_n\}$ $occur(pf_i) = (\bigcap_{pf_i \in pr_j} pr_j) \cap (\bigcap_{pf_i \notin pr_j} \overline{pr_j})$ $= \{pf_1, ..., pf_n\}$ *forms an AND-cluster.*

The first part of $occur(pf_i)$ is the set of probabilistic facts that occur in each proof in which pf_i occurs. The second part is the set of probabilistic facts that do not occur in proofs that do not contain pf. We use $\overline{pr_j}$ to denote the complement of the set pr_j with respect to the set of the probabilistic facts in all proofs. The first set is a possible AND-cluster for pf_i but it might also contain probabilistic facts that occur in proofs that do not contain pf_i. In order to exclude the latter ones, the possible AND-cluster has to be restricted to probabilistic facts that only occur in proofs containing pf_i.

4.1 The Book Marking Algorithm

Based on Lemma 1, the Book Marking algorithm in Table 1 deals with all the proofs one by one and ensures that for all probabilistic facts pf_i seen by the algorithm so far $occur(pf_i)$ is computed. The algorithm encodes a proof by a bit string. We order the probabilistic facts by their chronological appearance in the proofs. The i^{th} probabilistic fact is denoted by $pf(i)$. The i^{th} bit encodes whether the probabilistic fact $pf(i)$ is used in the proof. We refer to the bitstring as the **occurrence number** (ON) of the proof.

We use a two dimensional matrix (MA) of bits to represent the AND-clusters. Row k corresponds to the probabilistic fact $pf(k)$ and represents $occur(pf(k))$. Column l represents the probabilistic fact $pf(l)$. The element l of a row k indicates whether $pf(l)$ forms an AND-cluster with $pf(k)$. This matrix grows incrementally as we deal with the proofs one by one and the size of each dimension is equal to the number of different, already seen probabilistic facts.

Dealing with a new proof involves computing ON and then computing its impact on the AND-clusters already in MA according to Lemma 1 using the following three operations:

1. for each previously seen probabilistic fact i which appears in this proof, we compute $MA[i] = MA[i] \wedge ON$,
2. for each previously seen probabilistic fact i that does not occur in this proof, we compute $MA[i] = MA[i] \wedge \neg ON$,
3. we grow MA to include AND-clusters for the probabilistic facts that were not seen before.

After all proofs of the DNF have been dealt with, all the rows in MA with more than one active bit (i.e. set to 1) represent an AND-cluster.

```
bookmarking(DNF) {
  for each Proof in DNF {
    ON = bit_encode(Proof)
    for (i = 0; i < MatrixSize; i++) {
      if (2 ^ i & ON) > 0 then
      \\ Old row of pf in ON - operation 1
      MA[i] = MA[i] & ON
      else
      \\ Old row of pf not in ON - operation 2
      MA[i] = MA[i] & neg(ON)
    }
    for (i = MatrixSize; i < ListLength; i++) {
      \\ Add a new row - operation 3
      MA[i] = neg(2 ^ MatrixSize - 1) & ON
    }
    MatrixSize = ListLength
  }
}
```

Table 1. The Book Marking algorithm.

4.2 An Example of the Book Marking Algorithm

As an example for the Book Marking algorithm we use the proofs of `path(1,3)`: {x0, x2}, {x0, x3, x7}, {x1, x4, x5, x2}, {x1, x4, x5, x3, x7}. Each row of Table 2 corresponds to a single proof (PR), and has the probabilistic fact order (OL), the occurrence number (ON) and the matrix (MA).

For the first proof the algorithm uses the order $x0 < x2$ to compute 11 as the occurrence number of the proof. As initially the matrix MA is empty, operation 3 uses the ON to construct a MA with two rows and two columns with all bits activated.

For the second proof the algorithm adds $x3$ and $x7$ to the order which becomes $x0 < x2 < x3 < x7$. The algorithm computes $ON = 1101$; note that we are reading the bitstrings from right to left. Operation 1 computes the conjunction of 1101 with the row of $x0$: $11 \wedge 1101 = 0011 \wedge 1101 = 0001$. This operation sets the bit corresponding to $x2$ to 0 as $x0$ and $x2$ are no longer an AND-cluster. For the row of $x2$, operation 2 computes $11 \wedge neg(1101) = 0011 \wedge 0010 = 0010$

and sets the bit for the probabilistic fact $x0$ to 0. Finally, the algorithm extends MA by two new rows and columns for the probabilistic facts $x3$ and $x7$ with as values of the rows $neg(11) \wedge 1101 = 1100 \wedge 1101 = 1100$. Note that also the existing rows are expanded with new columns set to 0.

Proof(PR)	= x0, x2
Order List(OL)	= [x0, x2]
Occurrence Number (ON)	= 11 = 3
Matrix(MA)	= [3, 3]

1	1	x2
1	1	x0
x2	x0	

PR	= x0, x3, x7
OL	= [x0, x2, x3, x7]
ON	= 1101 = 13
MA	= [3 ∧ 13, 3 ∧ neg(13), neg(3) ∧ 13, neg(3) ∧ 13]
MA	= [1, 2, 12, 12]

1	1	0	0	x7
1	1	0	0	x3
0	0	1	0	x2
0	0	0	1	x0
x7	x3	x2	x0	

PR	= x1, x4, x5, x2
OL	= [x0, x2, x3, x7, x1, x4, x5]
ON	= 1110010 = 114
MA	= [1 ∧ neg(114), 2 ∧ 114, 12 ∧ neg(114), 12 ∧ neg(114), neg(15) ∧ 114, neg(15) ∧ 114, neg(15) ∧ 114]
MA	= [1, 2, 12, 12, 112, 112, 112]

1	1	1	0	0	0	0	x5
1	1	1	0	0	0	0	x4
1	1	1	0	0	0	0	x1
0	0	0	1	1	0	0	x7
0	0	0	1	1	0	0	x3
0	0	0	0	0	1	0	x2
0	0	0	0	0	0	1	x0
x5	x4	x1	x7	x3	x2	x0	

PR	= x1, x4, x5, x3, x7
OL	= [x0, x2, x3, x7, x1, x4, x5]
ON	= 1111100 = 124
MA	= [1 ∧ neg(124), 2 ∧ neg(124), 12 ∧ 124, 12 ∧ 124, 112 ∧ 124, 112 ∧ 124, 112 ∧ 124]
MA	= [1, 2, 12, 12, 112, 112, 112]

1	1	1	0	0	0	0	x5
1	1	1	0	0	0	0	x4
1	1	1	0	0	0	0	x1
0	0	0	1	1	0	0	x7
0	0	0	1	1	0	0	x3
0	0	0	0	0	1	0	x2
0	0	0	0	0	0	1	x0
x5	x4	x1	x7	x3	x2	x0	

Table 2. Book Marking algorithm example.

When all proofs are dealt with, the Book Marking Algorithm has found two different AND-clusters, namely $\{x1, x4, x5\}$ and $\{x3, x7\}$. Without variable compression, ProbLog generates the ROBDD of Figure 2c, which has a size in between the sizes of the other two ROBDDs in Figure 2. After compressing the variables of the AND-clusters to a representative variable $x1, 4, 5$ with $P(x1, 4, 5) = 0.252$ and $x3, 7$ with $P(x3, 7) = 0.32$, we get the compressed proofs: {x0, x2}, {x0, x3,7}, {x1,4,5, x2}, {x1,4,5, x3,7}. For the compressed proofs, ProbLog generates the ROBBD of Figure 3b.

The algorithm as presented here only tackles proofs that contains either positive or negative occurrences of each probabilistic fact and not both. If in one proof a probabilistic fact is positive and in an other is negative, this probabilistic fact does not form an AND-cluster.

5 Experiments for AND-clusters

We implemented the variable compression method using only AND-clusters within ProbLog. To judge the practicality and the impact we use ProbLog benchmarks that discover links in real biological networks [11]. Graphs model probabilistic links between concepts such as genes, proteins, etc.. The first benchmark consists of a graph of concepts related to the Alzheimer disease that has 23060 edges; because of the size, inference for this graph soon becomes intractable. We query for the existence of a path between two given nodes, to control the problem size we limit the maximum path length. For the second benchmark, we take the experiments (the same data sets and the same queries) from [1]. All the graphs are fragments of the same network [11]. The experiments should give answers to the following questions:

1. What is the compression ratio in a real life data set?
2. How does compression improve the performance of generating a ROBDD?
3. In which cases is the variable compression beneficial?

The default setting of ProbLog is to use CUDD's [13] group sifting [14] dynamic reordering during ROBDD generation. CUDD uses the following memory-time trade-off. It starts by consuming memory without reordering the variables, once the memory usage passes a threshold, it starts reordering the variables and as a consequence it consumes time. While CUDD is implemented in C, our Book Marking algorithm is implemented in Yap Prolog [15].

When we increase the problem size for the first benchmark, we see that the ROBDD generation time is the limiting factor. We executed three different queries with a timeout of 1 hour for the ROBDD generation. Each query was executed 5 times and we present the averaged times for the ROBDD generation. Table 3 presents the comparison of executing the queries with four different settings. The first and second column use the dynamic reordering strategy; the third and the fourth use the order in which probabilistic facts appear in the proofs as a static ordering; the first and third column use variable compression of AND-clusters. Our experiments confirm that for big ProbLog problems dynamic reordering performs better than static ordering. Variable compression improves the ROBDD generation times and has the expected effect both for dynamic and static orderings.

The second part of Table 3 presents the compression statistics which are independent of the reordering method: the time to do variable compression, the number of AND-clusters found, the number of variables before compression (ovars), the number of variables after compression (cvars), and finally the variable compression ratio[6]. We note that the time cost for doing the variable compression is by far less than the time gained during ROBDD generation. More importantly, the time for finding the AND-clusters is polynomial (as shown in the next section), while that for ROBDD generation is exponential. Because our

[6] Ratio = (ovars - cvars) / ovars

constant costs are relatively high, we notice that in small problems variable compression needs more time than we gain during ROBDD generation, but those problems are solved very fast either way. The benefit of variable compressing is far more significant for larger problem sizes.

	Path Length	Reordering Compressed	Reordering only	Static Compressed	Static	Compression statistics time	clusters	ovars	cvars	ratio
	8	4	4	5	5	7	11	34	23	32%
	9	51	97	7	9	22	17	91	71	22%
(a)	10	153	297	10	12	32	25	137	110	20%
	11	24,830	90,529	*	*	336	76	337	254	25%
	12	3,083,750	-	-	-	835	92	479	378	21%
	8	5	4	4	5	5	7	26	17	35%
(b)	9	282	417	24,904	47,000	72	49	170	119	30%
	10	1,035	1970	*	*	91	53	226	169	25%
	11	1,019,588	-	-	-	966	104	528	410	22%
	4	4	4	4	4	0	3	13	10	23%
(c)	5	95	246	18	23	64	42	135	91	33%
	6	224	497	74	122	33	45	180	131	27%
	7	58,917	2,488,793	*	*	385	92	455	350	23%

Table 3. First benchmark results. The reported times are in milliseconds. Longer path lengths timeout. A - indicates a timeout and * that the system runs out of memory.

In order to confirm the positive results for the compression ratio and the better performance of the ROBDD generation, we use the larger set of experiments of our second benchmark. We study the impact of variable compression in combination with dynamic reordering as it was confirmed to be the better option for ProbLog. In this benchmark, all the queries can be computed without variable compression. The behaviour of queries is diverse, as some spent most of the time in the ROBDD generation and others in SLD-resolution. Among the 360 queries of [1], 100 queries do not use any probabilistic facts. We divided the other 260 queries in 3 groups: the first group contains 92 queries that generate tiny ROBDDs with less than 20 variables; the second group contains 152 queries that generate small ROBDDs with 20 or more variables but less than 100; and finally the third group contains the queries that generated relatively big ROBDDs with more than 100 variables.

For the 'Tiny' group we obtain an average compression ratio of 42%. Their ROBDD generation times and the variable compression times are too small to draw any conclusions. For the other two groups, we compute averages for each group and for both groups together. The results are in Table 4. We give the variable compression ratio, the time gain realised for the ROBDD generation, and the variable compression time relative to the SLD resolution time.

While the results might be specific for the application, they confirm the actual presence of AND-clusters in real world datasets. In this real dataset we

Query Group	ROBDD Comp. Ratio	ROBDD Gen. Time Gain	Compression Time Ratio
Small	$(28 \pm 11)\%$	$(40 \pm 36)\%$	$(26 \pm 41)\%$
Big	$(27 \pm 5)\%$	$(47 \pm 23)\%$	$(69 \pm 107)\%$
All	$(28 \pm 10)\%$	$(41 \pm 36)\%$	$(32 \pm 53)\%$

Table 4. Averaged results and standard deviation. Where Comp. Ratio = (number of variables before compression - number of variables after compression) / number of variables before compression, ROBDD Gen. Time Gain = (ROBDD time without compression - ROBDD time with compression) / ROBDD time without compression and Compression Time Ratio= (SLD time with book marking algorithm - SLD time without) / SLD time without.

encounter a compression ratio that ranges from 7% to 61% with an average of 28%. The compression ratio results are similar to the ones of the first benchmark.

In the 'Small query' group 44% of the queries have a small number of variables and they do not need variable reordering neither before nor after compression: their time gain is near 0. Most of the other queries in the 'Small query' group need reordering before and no reordering after compression, so they have a huge time gain up to 87%. On average we end up with a gain of 40%.

For the 'Big query' group the average gain is larger namely 47%, but the variation is less as all the queries need reordering before and after compression. Here the gain comes from having less variables that have to be dealt with during the reordering by the state-of-the-art tool.

Comparing the variable compression time with the SLD-resolution time shows that the former is smaller than the latter, but its cost is relatively higher for the 'Big' queries.

Our experiments[7] yield promising results, answering our initial questions by showing that there is in real life ProbLog applications a role for variable compression as it improves significantly the performance of the ROBDD generation.

6 Complexity Analysis

The Book Marking algorithm in Table 1 has a worst case complexity of $O(M \cdot N^2)$ where M is the number of proofs seen and N is the number of different probabilistic facts; usually for ProbLog applications $M >> N$. For all M proofs, we do a bitwise encoding, and then we modify the matrix MA.

By using an indexed table, the encoding of a proof is done in $O(N)$ time and requires $O(N)$ space to remember the index for each probabilistic fact encountered. Each proof needs to modify MA which is an NxN bit table. Activating or deactivating a bit in the table is done in constant time, but in the worst case all bits needs to be processed resulting in $O(N^2)$ operations. So, the total time

[7] For our experiments we used an Intel^R Core^{TM}2 Duo CPU at 3.00GHz with 2GB of RAM memory running Ubuntu 8.04.2 Linux.

complexity is $O(M \cdot (N + N^2)) = O(M \cdot N^2)$ and the total space complexity $O(N + N^2) = O(N^2)$.

One can take advantage of the symmetry and other properties of the NxN bit matrix MA to avoid some computations. These optimisations reduce the constant times rather than the complexity. One such optimisation is that we use arbitrary precision integers to represent each row of MA.

7 Related Work and Conclusions

We exploit regularities, AND-clusters and OR-clusters, observed in ROBDDs to improve the generation of ROBDDs for DNFs in ProbLog. Variable compression based on these clusters reduces the number of variables in the DNFs. This results in smaller ROBDDs, whose generation uses less time and memory, and as such we can deal with ProbLog queries that used to cause timeouts. Our method is a pre-processing step that detects clusters of Boolean variables. Taking into account the probabilistic setting, variable compression is feasible and can be followed by any other variable ordering heuristic. For other applications, one might be able to find different meaningful compressions or one might just use our clusters as input to existing variable ordering heuristics.

Variable ordering heuristics also exploit structural properties of the problem modelled by the ROBDD such as connected variables [16, 14]. Heuristics designed for one application area might perform poorly in another context [17]. We are not aware of variable ordering heuristics to be used in a probabilistic context.

Hintsanen [18] argues that structural properties are important for finding the most reliable subgraph. He calculates the probability of subgraphs connecting two nodes and search for the subgraph with the maximum probability. The paper identifies as a special case the series-parallel subgraphs for which they can compute the probability polynomially. These series-parallel subgraphs have similarities with our AND/OR-clusters.

We have presented a polynomial algorithm for detecting the AND-clusters and we have obtained promising results for an application using a real database. For ProbLog the best results are obtained by combining AND-cluster variable compression with the group sifting dynamic variable ordering of CUDD. By using variable compression we managed to answer more queries. We showed that AND-cluster based variable compression is beneficial for more complex ROBDD.

For a future implementation of the Book Marking algorithm, C would be a better choice than Prolog both for time efficiency as for space. This would reduce many hidden constant costs of Prolog and would also save Prolog garbage collector executions. It is worth noting that the AND-clusters could be computed in parallel with the SLD-resolution.

In addition to the technical improvements, a challenging task is to investigate how we can take advantage of OR-clusters and compress the ROBDDs even more. Finally, the goal would be to generalise the method and to be able to compress repeated structures in the ROBDD. The size of the ROBDDs is one of the limits

that is currently reached when executing ProbLog programs. We think that an approach based on variable compression can push this limit.

8 Acknowledgements

We want to thank Bart Demoen and Angelika Kimmig for the valuable discussions and comments. This research is supported by: GOA/08/008 "Probabilistic Logic Learning".

References

1. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: Proceedings of IJCAI. (2007) 2462–2467
2. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of ProbLog programs. In: Proceedings of ICLP. (2008) 175–189
3. Akers, S.B.: Binary decision diagrams. IEEE Trans. Computers **27**(6) (1978) 509–516
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986) 677–691
5. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing **8**(3) (1979) 410–421
6. Rauzy, A., Châtelet, E., Dutuit, Y., Bérenguer, C.: A practical comparison of methods to assess sum-of-products. Reliab Eng Syst Safe **79**(1) (2003) 33 – 42
7. Fujita, M., Fujisawa, H., Kawato, M.: Evaluation and improvements of boolean comparison method based on binary decision diagrams. In: Proceedings of ICCAD. (1988) 2–5
8. Malik, S., Wang, A., Brayton, R., Sangionvanni-Vincentelli, A.: Logic verification using binary decision diagrams in a logic synthesis environment. In: Proceedings of ICCAD. (1988) 6–9
9. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Proceedings of ICCAD. (1993) 42–47
10. Somenzi, F.: Efficient manipulation of decision diagrams. STTT **3**(2) (2001) 171–181
11. Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., Toivonen, H.: Link discovery in graphs derived from biological databases. In: Proceedings of DILS. (2006) 35–49
12. Maier, D.: The complexity of some problems on subsequences and supersequences. ACM **25**(2) (1978) 322–336
13. Somenzi, F.: CUDD: Colorado university decision diagram package release 2.4.1 (2005) http://vlsi.colorado.edu/~fabio/CUDD/.
14. Panda, S., Somenzi, F.: Who are the variables in your neighborhood. In: Proceedings of ICCAD. (1995) 74–77
15. Santos Costa, V., Damas, L., Reis, R., Azevedo, R.: YAP User's Manual. (2002) http://www.ncc.up.pt/~vsc/Yap.
16. Aloul, F.A., Markov, I.L., Sakallah, K.A.: Faster SAT and smaller BDDs via common function structure. In: Proceedings of ICCAD. (2001) 443–448
17. Narodytska, N., Walsh, T.: Constraint and variable ordering heuristics for compiling configuration problems. In: Proceedings of IJCAI. (2007) 149–154
18. Hintsanen, P.: The most reliable subgraph problem. In: Proceedings of PKDD. (2007) 471–478