HAL open science

Structured Value Elimination with D-Separation Analysis

Lionel Torti, Pierre-Henri Wuillemin

▶ To cite this version:

Lionel Torti, Pierre-Henri Wuillemin. Structured Value Elimination with D-Separation Analysis. International Florida Artificial Intelligence Research Society Conference, May 2010, Daytona Beach, United States. pp.122-127. hal-00627846

HAL Id: hal-00627846
https://hal.science/hal-00627846v1

Submitted on 29 Sep 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Structured Value Elimination with D-Separation Analysis

Lionel Torti and **Pierre-Henri Wuillemin**
104, avenue du Président Kennedy
75016 Paris
France

Abstract

In the last ten years, new models based on Bayesian Networks (BN) emerged to handle large and complex systems. These new models can be divided in two: the unification with First Order Logic and uncertainty (Markov Logic Networks, Bayesian Logic) and Knowledge Base Construction Models (Probabilistic Relational Models, Multy-Entity Bayesian Networks, Relational Bayesian Networks). SKOOB, a consortium of researchers and engineers in risk management, focuses on Probabilistic Relational Models (PRM). Inference in such models is a much more difficult task than in BN. Structured Value Elimination (SVE) is the state-of-the-art algorithm for PRM models. In this paper, we propose an enhancement of SVE based on a well known complexity reduction technique from BN. We show how to integrate a d-separation analysis in SVE and how this leads to important improvements for the inference task.

Introduction

The need to represent uncertainty is a common issue in Artificial Intelligence and has been the main topic of a considerable amount of research. Bayesian Networks (BN) are a considered formalism for reasoning under uncertainty. Introduced twenty years ago, (Pearl 1988) and (Koller and Friedman 2009) they still are used for a great number of applications in domains such as automatic diagnostics, reliability and risk management (SKOOB 2008). The success of BN in industrial applications creates a need of tractable large scale complex systems. However BN have several flaws when considering specification of large scale networks which renders creation, maintenance and exploitation difficult or even impossible.

This issue is a partial cause of the recent emergence of new models, commonly called First Order Probabilistic Models (FOPM), which can be divided in two: the unification of First Order Logic and uncertainty (Markov Logic Networks, Bayesian Logic) and Knowledge Base Construction Models (Probabilistic Relational Models, Multi-Entity Bayesian Networks, Relational Bayesian Networks). The first ones are non oriented models (Markov Networks) in opposition to the second ones which are oriented (Bayesian Networks). While these models share a common theoretical background, their formalism may vary considerably.

In this paper we propose an enhancement of the state-of-the-art inference algorithm on Probabilistic Relational Models (PRM). PRM (Pfeffer 2000) are an extension of the BN formalism which includes the Object Oriented paradigm and well suited for specification, maintenance and exploitation of large scale complex systems. PRM can be seen as extensions of Object Oriented Bayesian Networks (Koller and Pfeffer 1997) and (Bangsö and Wuillemin 2000). While non oriented models have recent contributions regarding inference algorithms (Getoor and Taskar 2007), oriented models have very few propositions: Structured Value Elimination (Pfeffer 2000) and ground BN generation on which classic BN inference algorithm can be used (Pfeffer 2000) and (Laskey 2008).

The algorithm presented in this paper, Structured Value Elimination with d-separation analysis (SVED), is an amelioration of the Structured Value Elimination (SVE) algorithm (Pfeffer 2000) in which we have included a d-separation analysis of the PRM. While SVE exploits the structural information encoded in a PRM, d-separation helps reducing the necessary amount of information required to respond to a given query. D-separation analysis is a well known method yet rarely mentioned, despite its effectiveness in speeding up inference. The BayesBall algorithm (Shachter 1998) presents an intuitive and effective way to exploit d-separation in a BN and, as we will see, it can be easily used with the PRM formalism.

In this paper we will give a short definition of PRM, then we will recall the specificities of the SVE algorithm and present our contribution which will be validated by experimental results.

Probabilistic Relational Models

We give here a minimal set of definitions of the different elements of a PRM. For a more detailed presentation of PRM we suggest the following papers: (Pfeffer 2000) and (Getoor and Taskar 2007).

PRM are used to exploit the Object Oriented paradigm to produce large scale bayesian networks. The main intuition is to build a system with a set of BN fragments (classes) which are repeated (instances instantiations) in the network (system or relational skeleton). Relations of encapsulation

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

can be defined between classes and is used to hierarchically decompose a system in several subsystems.

- A *class* X is defined by a Directed Acyclic Graph (DAG) over a set of attributes (or random variables) $\mathcal{A}(X)$, references (or slots) $\mathcal{R}(X)$ and a probability law over $\mathcal{A}(X)$. We note $\mathcal{X}$ the set of classes of a PRM.

A class defines a family of objects sharing the same properties: DAG, attributes and references. The DAG coupled with the attributes defines the BN fragment. References are used to define dependencies between classes and, by extension, between instances. Since we want to represent probability dependencies between attributes of different classes with respect of the relations induced by the references, we need to define the notion of reference chains.

- We note $X.A$ an attribute A of a class X and we represent it by a node in X's DAG. An attribute represents a random variable and is associated with a conditional probability table defined over A and his parents $\Pi(A)$. We note $\mathcal{V}(A)$ the set of possible values taken by A.
- We note $X.\rho$ a *reference* ρ between two classes X and Y. The relation can be binary (simple) or n-ary (complex). For each reference ρ of X, we say that its *domain* is $Dom[\rho] = X$ and its *range* is $Range[\rho] = Y$.
- A *reference chain*, or *slot chain*, $\rho_1, \rho_2, \ldots, \rho_i$ is a sequence of reference where $Range[\rho_k] = Dom[\rho_{k+1}]$. A reference chain is simple if $\forall i, \rho_i$ is simple and is complex if $\exists i$ such as ρ_i is complex.

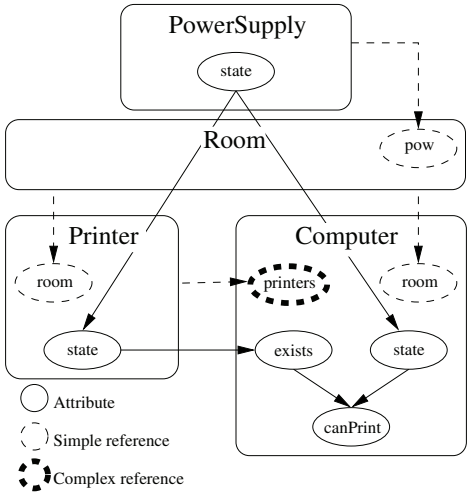


Figure 1: A Class Dependency Graph

Figure 1 illustrates the precedent notions. It represents a diagnosis problem in a local network between printers and computers. Each `Printer` and `Computer` are placed in a `Room` and each `Room` use one `PowerSupply`. There are several computers and printers per room and each computer is connected to zero, one or more printers. A peculiar attribute is `Computer.exists` which is called an aggregator: the reference `Computer.printers` is a complex reference which implies that `Computer.exists` have an unknown number of parents at class level. This is an issue since the classical method to encode probabilities in a BN is the use of Conditional Probability Tables (CPT) which implies the number of parents to be known. If we wanted to specify the CPT of `Computer.exists` it would be necessary to either define a family of CPTs at class level or specify the CPT at instance level for each instantiation of the `Computer` class. Both solutions are cumbersome: it is impossible to specify a family for all possible number of parents and specifying the CPT at instance level is unwanted since we want to separate the semantics of the instantiation.

- An *aggregator* is a deterministic functions over a set of attributes $\{x_i.A\}$ such as $\forall i, x_i \in \mathcal{I}(X)$.

By specifying the family of such functions enabled in the framework (exits, forall, mean, min, max, ...) it is possible to use deterministic CPTs to aggregate the information encode in the set $\{x_i.A\}$. Since these functions are deterministic it is possible to automatically generate the CPT when the aggregator is instantiated. This make them not differentiable from classic attributes at instance level. This solution is less expressive than the pseudo code used in MEBN (Laskey 2008), but is much simpler to use and implement. It is also generally sufficient for most authoring problems.

We finish with the definition of a system (or relational skeleton) and of the grounded BN of a system. We consider in this paper closed world systems, in opposition of open world systems, a closed world system requires to specify all instances and relations composing it. In an open world system, instances which are required but undeclared, are created automatically and added to the system.

- An *instance* x is the use of a given class X in a system σ_s. There can be more than one instantiation of a given class. We note $\mathcal{I}(X)$ the set of all instantiations of X, $x.A$ the instantiation of $X.A \in \mathcal{A}(X)$ in x and $x.\rho$ the instantiation of $X.\rho \in \mathcal{R}(X)$.
- A *system* (or relational skeleton) is a set of instances $\mathcal{I}$ with every reference of an instance $x \in \mathcal{I}$ correctly defined, i.e. $\forall X \in \mathcal{X}, \forall x \in \mathcal{I}(X), \forall \rho \in \mathcal{R}(x), \exists y \in \mathcal{I}(range[X.\rho])$ and $range[x.\rho] = y$.
- A *grounded Bayesian Network* is the BN equivalent of a system. It is built straightforwardly by creating a node for each attribute and aggregator and adding arcs between nodes to respect the instances relations. An algorithm linear in time for generating such grounded BN can be found in (Pfeffer 2000).

There are several other notions that we will not detail here such as: attribute typing, inheritance, structural and existential uncertainties. Even if they are important in the PRM formalism they are not exploited by the different inference algorithms presented in this paper.

SVE : Structured Value Elimination

Structured Value Elimination (SVE) (Pfeffer 2000) is an extension of the classical inference algorithm Value Elimination (VE) (Zhang and Poole 1994) in the PRM formalism. The VE algorithm places in a pool the CPT of a BN's nodes

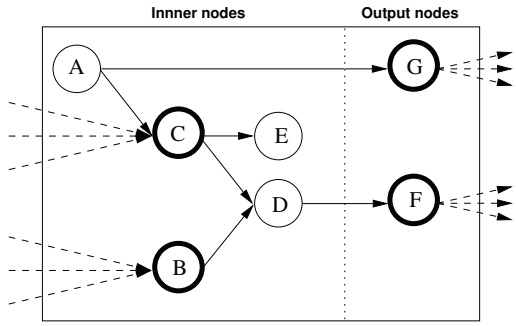


Figure 2: Inner and outer nodes in a class

and uses an order on these nodes to eliminate the random variables one by one from the pool. When all the variables except the query are eliminated, the pool contains the query's marginal in a unnormalized potential. Exact inference in a BN is NP-Hard (Cooper 1988) and the order used by VE has a major impact on the algorithm's performance.

SVE exploits the structural information encoded in the classes to avoid redundant computations. However the algorithm presented in (Pfeffer 2000) is designed to exploit open world systems and in our case we want to infer only over closed world systems (which are more often encountered in risk management and reliability). Nevertheless, this difference does not change dramatically the concepts behind SVE.

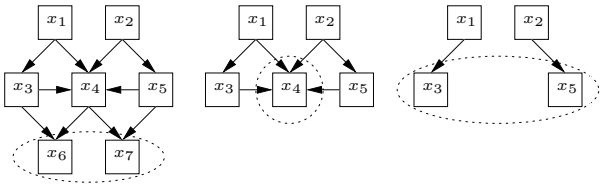
Eliminated Nodes	Potentials
$\{\}$	$\{ P(A), P(G\|A), P(B\|\Pi(B)), P(C\|A,\Pi(C)), P(D\|B,C), P(E\|C), P(F\|D) \}$
$\{A\}$	$\{ \Phi_1(C,G,\Pi(C)), P(B\|\Pi(B)), P(D\|B,C), P(E\|C), P(F\|D) \}$
$\{A,B\}$	$\{ \Phi_1(C,G,\Pi(C)), P(E\|C), \Phi_2(C,D,\Pi(B), P(F\|D) \}$
$\{A,B,C\}$	$\{ \Phi_3(D,E,G,\Pi(B),\Pi(C)), P(F\|D) \}$
$\{A,B,C,D\}$	$\{ \Phi_4(E,F,G,\Pi(B),\Pi(C)) \}$
$\{A,B,C,D,E\}$	$\{ \Phi_5(F,G,\Pi(B),\Pi(C)) \}$

Table 1: Elimination of inner nodes in the figure 2.

To exploit structural information we must differentiate two kinds of nodes: the inner nodes and the output nodes. Inner nodes are nodes with children exclusively defined in the same class, while output nodes have at least one child defined in another class. In the figure 2 A, B, C, D and E are inner nodes. F and G are output nodes. The table 1 shows the resulting potentials obtained after elimination of the inner nodes in our example.

Once all inner nodes are eliminated the resulting potentials are over output nodes and the set of referenced parents. SVE exploits this by eliminating at class level inner nodes and reusing the resulting potentials each time an instance of the given class is found in the system. There is however one exception of inner nodes which cannot be eliminated at class level: the aggregators. Since these node have an unknown number of parents at class level, it is impossible to used their CPT to proceed to any elimination anywhere else than instance level. There is a limitation to the reuse of the inner nodes elimination: in the presence of evidence over those nodes it is not possible to reuse the class level elimination, simply because it does not include the evidences in the computations. This renders necessary to eliminate the inner nodes of an instance with observations locally.

If applied to all instances in a system, this inner elimination builds a pool of potentials containing only output nodes and aggregators. To prevent a two step elimination, SVE uses a bottom-up elimination order to recursively eliminate instances. This gives the guarantee that before an instance is eliminated, all its child instances are eliminated. Thus making possible the elimination of its output nodes and consequently reducing the number of variables in the pool.



(a) Step 1. (b) Step 2. (c) Step 3.

Figure 3: Illustration of Bottom-Up Elimination.

Figure 3, where the different x_i represents instances of different classes, shows how a bottom-up elimination proceeds. If we suppose the queried variable resides in x_1, a recursive call is made over x_1's children: x_3 and x_4. The instance x_3 has two children, x_6 and x_4. Since x_6 is a leaf instance it can be eliminated and the recursion continues over x_4 which will end in a recursive call over x_7. The figure 3(b) shows the remaining instance after these two eliminations. The recursive elimination continues over x_4 until all instances different from x_1 are eliminated, making possible the extraction of the queried variable updated with evidences if any.

D-Separation analysis in PRM

As stated by (Shachter 1998), d-separation analysis (w.r.t the structure and the evidence) may induce another kind of optimisation: only a subset of the BN may be necessary in order to answer a specific request. D-separation exploits the graphical properties of BN to determine the requisite nodes necessary for computing the marginal of a queried variable given observations in the network.

The BayesBall algorithm (Shachter 1998) exploits d-separation to determine in a simple yet effective way the set of requisite nodes given a query and a set of observations. It uses an imaginary ball which bounces between the nodes of the BN's DAG, following the flow of information.

In this work, our aim is to adapt such algorithm to PRM. A d-separation analysis of a PRM will help reducing the number of computations, which can be seen as a low level exploitation of the structural information encoded in a PRM.

There are two possible approaches to exploit the Bayes-Ball algorithm in a PRM. A naive approach would be to ground the system and determine the requisite set using the BayesBall algorithm and a mapping of the BN's nodes with the attributes in the system. But unless using an intelligent implementation, which will be redundant with the PRM implementation, the memory cost of creating such grounded BN will grow rapidly.

The other approach consists in adapting the BayesBall algorithm to directly reason over a PRM. This can be done with very few intuitive modifications of how we represent classes and instances. The problem is that informations about attribute's parents and children are not encoded in the classes or instance's DAG and since BayesBall reasons over a BN's DAG we need to represent external dependencies between attributes of different instances. This can be done by adding two types of nodes in classes and instance's DAG: slot chains nodes and inverse slot chains nodes.

We have seen that PRM uses slot chains to create dependencies between attributes of different classes. An inverse slot chains is the reverse path defined by a given slot chains. If we look at the `Printer` class in figure 1 we see that the attribute `Printer.state` has one parent: `PowerSupply.state` referenced by the slot chain `Printer.room.pow.state`; one child: `Computer.exists` referenced by `Computer.printers.state` and its inverse slot chain of `Computer.printers.state` is `Printer.inv(printers).exists`. By adding a node for each slot chain and inverse slot chain, it is possible to use the BayesBall algorithm if we consider these nodes as attributes with no observation. When the ball arrives on a slot chain or inverse slot chain we simply make it follow the reference represented by the node.

SVED : Structured Value Elimination with D-Separation Analysis

The following algorithms describe the SVE with d-separation analysis. Algorithm 1 arguments are the instance (Q) and its attribute ($query$) for which we want to compute the marginal. The remaining argument is an empty set of potentials which will contain the marginal over the query at the end of the process. Downward Elimination (DnEl) and Upward Elimination (UpEl) are two procedures called respectively on child instances and parent instances.

At line 1 of the algorithm 1 there are two set of instances, $visited$ and $not_visited$, used respectively to prevent any redundant elimination of an instance and to delay the computation of the other ones.The need to delay the elimination of some instances is due to the bottom-up elimination used by SVE. When an instance i is eliminated the elimination order guaranties that DnEl was called on all child instances of i. However the parents instances of i have not been eliminated yet and it is necessary to keep track of them so they can be eliminated after all their child instances (line 13 of algorithm 2). This is realized by algorithm 1 at line 14 and by algorithm 3 at line 15.

To know if an instance i can call the DnEl procedure on an instance j, we use the notion of precedence between classes. Using the Class Dependency Graph it is simple to define an order between classes: for two classes C_1 and C_2 if an attribute in C_2 is an ancestor of at least one attribute in C_1, then C_1 precedes C_2. If both attributes in C_2 and in C_1 are ancestors of nodes in, respectively, C_1 and C_2 then an arbitrary choice is necessary. In such case line 3 in both algorithms 1, 2 and 3 guarantees computations soundness.

The call of the sub-procedure VE stands for a call of the Value Elimination algorithm over the requisite nodes of the current instance (line 8 of algorithm 1, line 8 of algorithms 2 and 3). When VE is given a class as argument it reuses any elimination already made (line 9 of algorithm 2 and 3). The Instance function returns the set of instances assigned to a given slot chain or inverse slot chain.

Algorithm 1: SVED

```
   Input: Q, query, pool
1  Set visited, not_visited;
2  visited = {Q} ∪ visited;
3  pool = Requisite(Q) ∪ pool;
4  for n ∈ Requisite(Q) do
5      if n is a inverse slot chain then
6          for i ∈ Instance(n) do
7              DnEl(i, pool, visited, not_visited);
8  VE(Q, query, Requisite(Q), pool);
9  for i ∈ not_visited do if Q precedes i then
10     DnEl(i, pool, visited, not_visited);
11 for n ∈ Requisite(Q) do
12     if n is slot chain then
13         for i ∈ Instance(n) do
14             UpEl(i, pool, visited, not_visited);
```

Algorithm 2: Downward Elimination (DnEl)

```
   Input: i, pool, not_visited, visited
1  if i ∉ visited then
2      visited = {i} ∪ visited;
3      pool = Requisite(i) ∪ pool;
4      for n ∈ Requisite(i) do
5          if n is an inverse slot chain then
6              for j ∈ Instance(n) do
7                  DnEl(i, pool, visited, not_visited);
8      if hasEvidence(i) then VE(i, Requisite(i), pool);
9      else VE(Class(i), Requisite(Class(i)), pool);
10     for n ∈ Requisite(i) do
11         if n is a slot chain then
12             for j ∈ Instance(n) do
13                 not_visited = {j} ∪ not_visited;
```

Algorithm 3: Upward Elimination (UpEl)

Input: i, $pool$, $not_visited$, $visited$

if $i \notin visited$ **then**
  $visited = \{i\} \cup visited$;
  $pool$ = Requisite(i) $\cup$ $pool$;
  for $n \in$ *Requisite(i)* **do**
    if n *is an inverse slot chain* **then**
      for $j \in$ *Instance(n)* **do**
        DnEl(j, $pool$, $visited$, $not_visited$);
  if *hasEvidence(i)* **then** VE(i, Requisite(i), $pool$);
  else VE(Class(i), Requisite(Class(i)), $pool$);
  for $j \in not_visited$ **do** **if** i *precedes* j **then**
    DnEl(j, $pool$, $visited$, $not_visited$);
  for $n \in$ *Requisite(Q)* **do**
    if n *is a slot chain* **then**
      for $i \in$ *Instance(n)* **do**
        UpEl(i, $pool$, $visited$, $not_visited$);

Experimental results

For the following experiments we have used different systems created from the Class Dependency Graph illustrated in figure 1. The examples we used to realize our experiments are generally composed of a fixed number of rooms (fifty in our case) and a random number of printers and computers per room. The number of computers have an incident on the size of the network, while the printers have a direct impact in the size of the biggest clique since every printers state are aggregated by each computer in the same room.

Figure 4 illustrates an experiment where we randomly observed a percentage of attributes in a system of 50 rooms, 10 printers and 40 computers (for a total of 6501 nodes). The queried attribute was randomly chosen. The curve of the figure 4 shows how much d-separation analysis is sensible to the set of observations and the queried node. The markov blanket of a variable d-separates it from the rest of the graph, which is shown when the percentage of observed nodes tends to 100 percent. When the percentage of observed nodes tends to 0 percent, the curves indicates that in the absence of information very few nodes are required. However these results are highly dependent of the system on which we obtained them and only indicates that we can expect d-separation to be more efficient in the presence of few or many observations. Noteworthy the whole PRM is almost never needed. This confirms the utility of such an analysis.

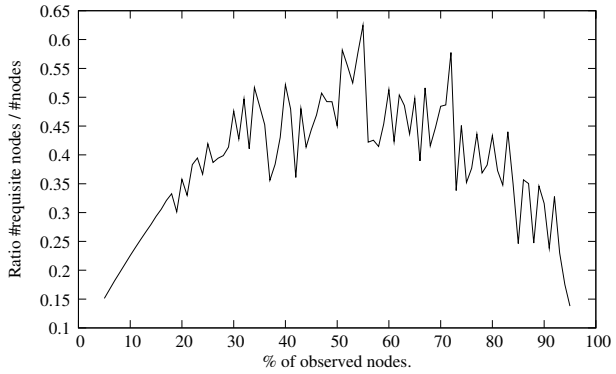


Figure 4: D-separation analysis with BayesBall.

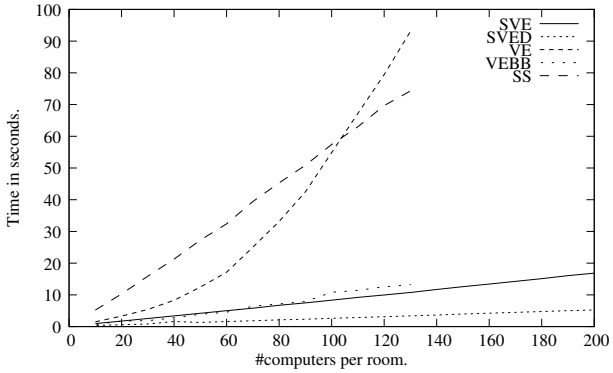


Figure 5: Structured and grounded inference.

Figure 5 exhibits the behaviour of several inference algorithms in the absence of observations. For this experiment we have raised the number of computers per room from 10 to 200, with 6 printers and 40 rooms (for a total of 1441 to 24241 nodes). The curves which stops after 130 computers represents classic inference algorithms used on the grounded BN of the tested system. The following algorithms on BN were used: Value Elimination (VE) (Zhang and Poole 1994), Shafer-Shenoy (SS) (Shenoy and Shafer 1990) and Value Elimination coupled with BayesBall (Shachter 1998). The results shows the limitation of reasoning on BN, since the computer used for the tests could not handle the size of the grounded BN. Furthermore SVE and SVED gives good results.

Figure 6 show the behaviour of SVE and SVED in the absence of observations (the SVED curve is close to 0). The system contained 40 rooms with 50 computers and printers varying from 1 to 10 (for a total of 6041 to 6401 nodes). The

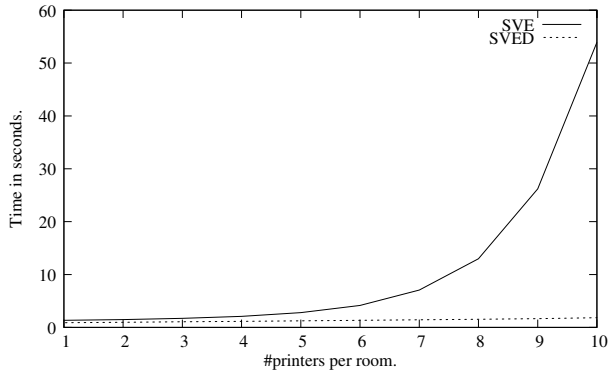


Figure 6: Comparaison between SVE and SVED.

curves clearly show the possible gain d-separation analysis can give to inference. However we must point out that the example used for these test gives very good performance to algorithms exploiting d-separation and they should be considered as best case situations. Although in the worst case situation, its easy to show that SVE and SVED do not differ neither in complexity nor in time.

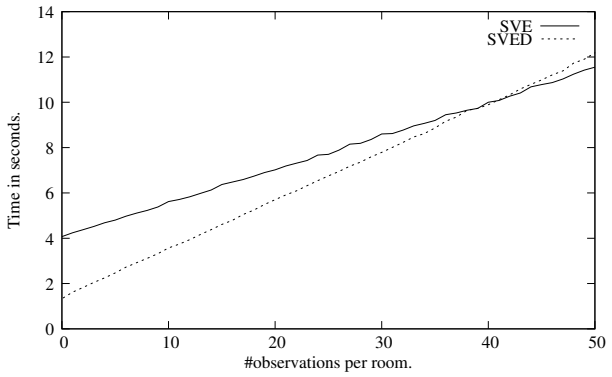


Figure 7: Inference performance under observations (6 printers per room).

Figure 7 shows the impact of observations on SVED and SVE in a system with 6 printers, 50 computers per room and 40 rooms. It gives the inference time when the power supply's state attribute is queried under different amounts of observations. The number of observed nodes in each room grows from 0 observed computers to 50. These results gives us the insights of SVED performances under heavy observations: we can expect slower inference, even slower then SVE in the worst case. The worst case for SVED would be systems with small cliques.

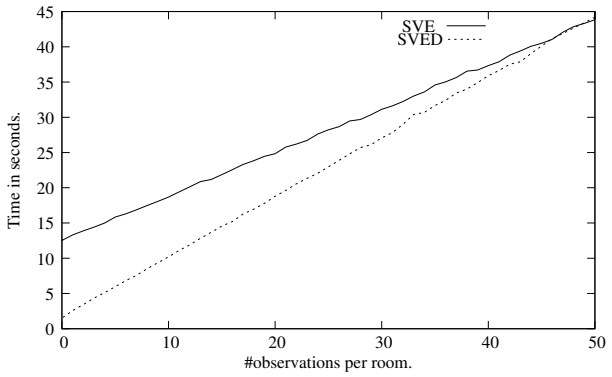


Figure 8: Inference performance under observations (8 printers per room).

Figure 8 represents results of a similar experiment has the one represented in figure 7. The difference reside in the number of printers per room, which was raised to 8. This illustrates the importance of the size of the largest clique in inference complexity. We can see that the cost of d-separation analysis is quickly negligible when inferring in systems with large cliques.

Conclusion

In this paper we propose to integrate the BayesBall algorithm in SVE. This simple modification may lead to a large reduction of complexity for inference in PRM. Furthermore, this work gives experimental results on inference in PRM, which is absent from the current state-of-the-art. These results confirm the usefulness of developing inference algorithms on PRM, thus we can say that exact inference in PRM is a far from closed domain since there are still specificities, such as inheritance or structural uncertainty, that could be exploited to speed up inference. Value Elimination is only one of many inference algorithms in Bayesian Networks and adapting such algorithms could prove useful.

Acknowledgments

This work has been supported by the DGA and has benefit comments, suggestions and ideas of the SKOOB consortium members.

References

Bangsö, O., and Wuillemin, P.-H. 2000. Top-down construction and repetitive structures representation in bayesian networks. In *Proceedings of the 13th Florida Artificial Intelligence Research Society Conference*, 282.

Cooper, G. F. 1988. Probabilistic inference using belief network is np-hard. Technical Report KSL-87-27, Medical Computer Science, Stanford University, Stanford, California.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

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

Koller, D., and Pfeffer, A. 1997. Object-oriented Bayesian networks. In *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, 302–313. Winner of the Best Student Paper Award.

Laskey, K. B. 2008. MEBN: A language for first-order bayesian knowledge bases. *Artificial Intelligence* 172:140–178.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pfeffer, A. J. 2000. *Probabilistic Reasoning for Complex Systems*. Ph.D. Dissertation, Stanford University.

Shachter, R. 1998. Bayes-ball: The rational pastime. In *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI)*, 480–487. Morgan Kaufmann.

Shenoy, P., and Shafer, G. 1990. Axioms for probability and belief-function propagation. In Shafer, G., and Pearl, J., eds., *Readings in Uncertain Reasoning*. San Mateo, CA: Morgan Kaufmann. 575–610.

SKOOB. 2008. `http://skoob.lip6.fr`.

Zhang, N., and Poole, D. 1994. A simple approach to bayesian network computation. In *10th Canadian Conference on Artificial Intelligence*, 16–22.