Patterns Discovery for Efficient Structured Probabilistic Inference

Lionel Torti, Christophe Gonzales, and Pierre-Henri Wuillemin

Université Pierre et Marie Curie,
Laboratoire d'Informatique de Paris 6,
75252 Paris Cedex 05, France,
firstname.lastname@lip6.fr

Abstract. In many domains where experts are the main source of knowledge, e.g., in reliability and risk management, a framework well suited for modeling, maintenance and exploitation of complex probabilistic systems is essential. In these domains, models usually define closed-world systems and result from the aggregation of multiple patterns repeated many times. Object Oriented-based Frameworks (OOF) such as Probabilistic Relational Models thus offer an effective way to represent such systems. OOFs define patterns as classes and substitute large Bayesian networks (BN) by graphs of instances of these classes. In this framework, Structured Inference avoids many computation redundancies by exploiting class knowledge, hence reducing BN inference times by orders of magnitude. However, to keep modeling and maintenance costs low, OOF classes often encode only generic situations. More complex situations, even those repeated many times, are only represented by combinations of instances. In this paper, we propose to determine such combination patterns and exploit them as classes to speed-up Structured Inference. We prove that determining an optimal set of patterns is NP-hard. We also provide an efficient algorithm to approximate this set and show numerical experiments that highlight its practical efficiency.

1 Introduction

Bayesian networks (BN) [19] are a valued framework for reasoning under uncertainty and their popularity stimulated the need for handling problems of ever increasing size. However BNs turn out to be inadequate for large scale real-world applications due to high design and maintenance costs [18, 20]. Indeed, defining a BN requires to specify explicitly probabilistic dependencies and conditional probabilities over the whole set of its random variables. This may lead to unrealistic modeling costs when dealing with complex systems. Furthermore, BN's design is static: any change in the topology of their graphical structure induces significant update costs.

Solving these problems has been the main concern of several BN extensions using the object-oriented paradigm [15, 18]. Besides, first-order logic extensions were proposed to offer more expressive power than the propositional framework offered by BNs [13, 14]. Learning being a critical problem when exploiting BNs over large knowledge bases, entity-relationship extensions were also proposed for relational learning [8, 10]. These extensions are all allegedly considered as First-Order Probabilistic Models (FOPM) or as Knowledge Based Construction Models.

During the last decade, the Probabilistic Graphical Model (PGM) community has worked actively on FOPMs and object-oriented models have been somewhat neglected: since the introduction of Object-Oriented Bayesian Networks [3, 15], the amount of contributions on object-oriented PGMs has actually been relatively small [1, 2, 8]. However, in many industrial areas, efficient frameworks for the construction of large-scale complex systems are strongly needed and, in domains like risk management or monitoring of complex industrial processes, this usually boils down to experts modeling large-scale BNs by aggregating hierarchically small network fragments repeated many times. In addition, all the relations between these fragments are usually fully specified, thus resulting in modeling "closed worlds". For these domains, object-oriented frameworks seem more suitable than first-order logic extensions. In particular, the "closed world" assumption strongly degrades the behavior of lifted inference in FOPM.

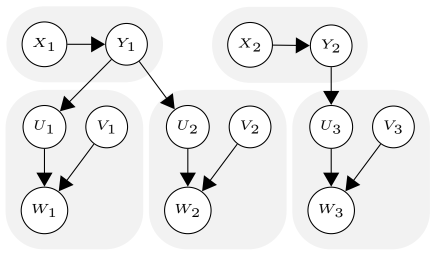
Object-oriented frameworks assume that many parts of a large BN are similar and can thus be described as instances of a generic class defined only once. This scheme induces low construction costs. In addition, maintenance costs are kept as low as possible since a modification in a class definition updates many areas of the BN at once. Furthermore, repetitions of structures in the BN (multiple instances of the same class) can speed-up inference by performing computations within classes, caching them and using the cache for all their instances. This process allows algorithms like *Structured Variable Elimination* (SVE) to outperform classical BN inference engines by orders of magnitude [21].

In this paper, we propose an enhancement of structured inference for Probabilistic Relational Models [7, 25]. In real world applications, instances are often combined and form patterns repeated many times throughout the network. By using a frequent subgraph pattern mining algorithm, it is possible to discover such combinations and exploit them to speed-up structured inference. However, mining optimally such patterns is time expensive. In this paper, we both provide a structured inference algorithm for PRMs exploiting patterns and a mining heuristic fast enough for efficient inference.

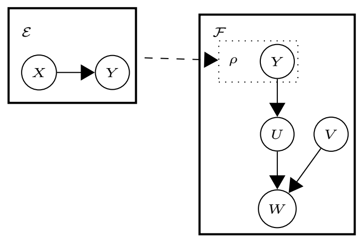
The paper is organized as follows: Section 2 recalls the basics of object-oriented frameworks using PRMs. Section 3 generalizes structured inference. Section 4 shows the complexity of mining patterns and provides an approximate algorithm for such mining. Experiments reported in Section 5 show the practical efficiency of our approach. Finally, concluding remarks are given in Section 6.

2 Description of PRMs

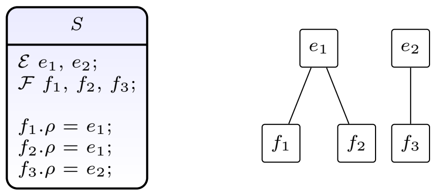
Using PRMs as an object-oriented framework can be surprising as they were first proposed for relational learning [5]. However, it is important to remember that PRMs are an extension of Object Oriented Bayesian Networks [21, 25] and, thus, they offer a sound object-oriented framework.



(a) A Bayesian network. The gray areas do not belong to the BN specification



(b) Two connected classes $\mathcal{E}$ and $\mathcal{F}$.



(c) System declaration and relational skeleton for the BN of Figure 1(a).

Fig. 1. Representation of a BN as a PRM: analysis of the BN reveals the use of two recurrent patterns (a), which are confined in two classes (b). Hence, a system equivalent to the BN may be built (c).

Due to a lack of space, we only present briefly and incompletely the PRM framework [7, 25]. Fig. 1.(a) shows a BN encoding relations between two different kinds of patterns (variables $\{X_i, Y_i\}$ and $\{U_j, V_j, W_j\}$). Variables whose names begin with the same letter share identical conditional probability tables (CPT). Object-oriented representations aim to abstract each pattern as a generic entity (a *class*) that encapsulates all the relations between the variables of the pattern. So, in Fig. 1.(b), class $\mathcal{E}$ encapsulates precisely variables X_i and Y_i as well as their probabilistic relations (arc (X_i, Y_i)) and their CPTs. The pattern of variables U_j, V_j, W_j cannot be directly encapsulated in a class since the CPTs of variables U_j are conditional to some variables Y_k (e.g., the CPT of U_3 is $P(U_3|Y_2)$ according to Fig. 1.(a)). Hence classes must have a mechanism allowing to reference variables outside themselves. In PRMs, this mechanism is called a *reference slot*. A reference slot ρ of a class $\mathcal{C}$ is a local name for another class $\mathcal{D}$ allowing $\mathcal{C}$ to access its variables. As shown in Fig. 1.(c), the original BN can then be built up from the PRM: it is sufficient to create two instances, say e_1 and e_2, of class $\mathcal{E}$ as well as three instances f_1, f_2, f_3 of $\mathcal{F}$ and connect them using one edge per reference slot.

Definition 1 (Class, Attribute, Reference slot). *In order to define classes, one needs the following cross-definitions:*

• *A* class *is a quadruple* $\langle \mathbf{A}(\mathcal{C}), \mathbf{R}(\mathcal{C}), G(\mathcal{C}), \mathbf{P}(\mathcal{C})\rangle$ *where:*

• $\mathbf{A}(\mathcal{C})$ *is a set of attributes. An attribute* $\mathcal{C}.X \in \mathbf{A}(\mathcal{C})$ *is a random variable.* $\mathcal{C}$ *is called the resident class of* X*.*

• $\mathbf{R}(\mathcal{C})$ *is a set of reference slots.* $\mathcal{C}.\rho \in \mathbf{R}(\mathcal{C})$ *is a surrogate for another class, say* $\mathcal{D}$*, giving access to all its attributes and reference slots from within* $\mathcal{C}$*. Range*(ρ) *denotes class* $\mathcal{D}$*. The set* $\overline{\mathbf{A}(\mathcal{C})}$ *of all the attributes reachable by way of reference slots or within* $\mathcal{C}$ *is called the closure of* $\mathcal{C}$*.*

• $G(\mathcal{C}) = (\overline{\mathbf{A}(\mathcal{C})}, E)$ *is a Directed Acyclic Graph (DAG) where* $E \subseteq \overline{\mathbf{A}(\mathcal{C})} \times \mathbf{A}(\mathcal{C})$ *: only the attributes of* $\mathcal{C}$ *have parents in this DAG.*

• $\mathbf{P}(\mathcal{C}) = \{P(X|\Pi_X), X \in \mathbf{A}(\mathcal{C})\}$ *is the set of CPTs of attributes* $X \in \mathbf{A}(\mathcal{C})$ *conditionally to their parents in* $G(\mathcal{C})$*.*

Classes are not meant to be used as is, but through instances. For example, a class may represent various failure odds of a cooling system in a nuclear power plant and, when modeling a given power plant, such class is instantiated for each occurrence of the cooling system in the whole plant.

Definition 2 (Instance, System). *Let* $\mathcal{B}$ *be a BN,*

• *An* instance c *of a class* $\mathcal{C}$ *is a subset of the random variables of* $\mathcal{B}$ *whose relations are described by* $\mathcal{C}$*.* $c.X$ *(resp.* $c.\rho$*) refers to the instantiation of* $\mathcal{C}.X \in \mathbf{A}(\mathcal{C})$ *(resp.* $\mathcal{C}.\rho \in \mathbf{R}(\mathcal{C})$*) in* c*. By abuse of notation, we denote the sets of such instantiations as* $\mathbf{A}(c)$ *and* $\mathbf{R}(c)$ *respectively.*

• *A system* S *is the representation of* $\mathcal{B}$ *in the PRM framework: it is a finite set of instances such that* $\forall i \in S, \forall \rho \in \mathbf{R}(i), \exists j \in S$ *such that Range*$(i.\rho) = j$ *and such that there is a one-to-one mapping between random variables of* $\mathcal{B}$ *and the set of all attributes declared in the instances of* S*.*

The graph representing instances by nodes and connections between range and resident instances by edges is called the relational skeleton of S*.*

Definition 2 enforces the "closed world" feature of systems, i.e., they are finite sets of instances with all reference slots properly defined. As mentioned in the introduction, this constraint is reasonable for complex systems of many domains. For instance, to reason on industrial milk fermenters, pipe connections need to be fully specified.

3 Structured Inference

Determining the probabilities of random variables given evidence is the most common query performed in probabilistic graphical models. There exists a wide range of inference algorithms to compute these distributions. They often rely in some way to a Variable Elimination scheme [4, 17]. The basic idea consists of marginalizing out random variables one by one from the joint distribution until there only remains the variables of interest. Dechter's Variable Elimination (VE) is representative of this class of algorithms. It first fills a pool of functions called *potentials* with the CPTs representing the decomposition of the joint distribution. Then, eliminating some variable X_j from the joint probability just amounts to extract from the pool all the potentials involving X_j, multiply them and sum-up the result over all the values of X_j, and insert back the resulting potential into the pool. Conditional probabilities $P(\mathbf{X}|\mathbf{e})$ are computed similarly by first adding to the pool some potentials representing the additional knowledge brought by evidence $\mathbf{e}$.

The above scheme is efficient and can be used in PRMs by applying it on their grounded BN. However, by processing random variables separately, VE is unable to exploit the structural repetitions in the graphical model to avoid computation redundancies. The aim of Structured Inference is to fill this gap [5, 21] and Object-Oriented frameworks provide a simple and effective way to achieve this goal. Indeed, consider an attribute A of a class $\mathcal{C}$ such that all of its children also belong to $\mathcal{C}$ and let $c_1, \ldots, c_k$ be some instances of $\mathcal{C}$ in which no attribute received any evidence. Then it is easy to see that eliminating attributes $A \in \mathbf{A}(c_i)$ in the grounded BN produces precisely the same computations for all the instances c_i, $i = 1 \ldots, k$. In this case, eliminating attribute A within class $\mathcal{C}$, i.e., *at class level*, and updating accordingly all the relevant instances before constructing the grounded BN avoids the redundancies involved by eliminating A in each c_i, i.e., *at instance level*. This process is called *Structured Inference* and the gain brought by this approach usually reduces computation times by orders of magnitude.

More Formally, an attribute $A \in \mathbf{A}(\mathcal{C})$ is called an *inner* or *internal* attribute if all of its children also belong to $\mathbf{A}(\mathcal{C})$, otherwise A is called an *outer* attribute. In addition, the attributes referenced in $\mathbf{R}(\mathcal{C})$ are called *non-resident*. For instance, in Fig. 1.b, attributes X, U, V, W are internal, Y is an outer attribute of class $\mathcal{E}$ and $\rho.Y$ is a non-resident attribute of $\mathcal{F}$. Class-level elimination corresponds to the elimination of all the inner attributes (using any inference algorithm). As such, it amounts to substitute the pool of potentials $\mathbf{P}(\mathcal{C})$ of class $\mathcal{C}$ defined over all of its inner, outer and non-resident attributes by a new set of potentials $\mathbf{P}'(\mathcal{C})$ defined only over the outer and non-resident attributes. The pool of potentials corresponding to any instance c of $\mathcal{C}$ is thus substituted by $\mathbf{P}'(c)$ if no inner attribute in c received any evidence, else it is kept to $\mathbf{P}(c) \cup \{\text{potentials(evidence)}\}$ (because evidence may induce different distributions from one instance to another).

4 PRM's Patterns Discovery

4.1 Problem and Complexity

Marginalizing-out internal nodes at *class level* is the key to Structured Inference efficiency as it reduces significantly redundant computations. However, not all redundancies can be identified by this scheme: let $\mathcal{C}, \mathcal{D}$ be two classes and let $X \in \mathbf{A}(\mathcal{C})$, $Y \in \mathbf{A}(\mathcal{D})$ be two attributes such that the only non-resident child of $\mathcal{C}.X$ is $\mathcal{D}.Y$. Then X cannot be eliminated at class level because it is not internal. However, if we consider a "new" class $\mathcal{F}$ defined by compound $(\mathcal{C}, \mathcal{D})$, attribute $\mathcal{F}.X$ is no longer an outer attribute since $Y \in \mathbf{A}(\mathcal{F})$. Hence, pairs of instances (c,d) of $\mathcal{C}$ and $\mathcal{D}$ that fit the definition of $\mathcal{F}$ can be considered as instances of $\mathcal{F}$ in which X is internal, thus eligible for class-level elimination. Note however that not all pairs (c,d) are necessarily eligible: in Fig. 1, pairs (e_1, f_1) and (e_1, f_2) cannot be both considered as instances of compound $(\mathcal{E}, \mathcal{F})$ as e_1 would be counted twice in the grounded BN. Pair (e_1, f_3) is neither eligible because there is no edge between e_1 and f_3 in the system. The key idea is that finding effective compounds/instance-class reassignments should speed-up Structured Inference since it increases the opportunities for class-level eliminations. It is most convenient to search them in the following graph:

Definition 3 (boundary graph). *A boundary graph is an undirected graph $\boldsymbol{\mathcal{BG}} = (\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{E}})$, where*

- *$\boldsymbol{\mathcal{I}}$ is a set of vertices representing instances;*
- *$\boldsymbol{\mathcal{E}} \subseteq \boldsymbol{\mathcal{I}} \times \boldsymbol{\mathcal{I}}$ is a set of edges such that $\exists (c,d) \in \boldsymbol{\mathcal{E}}$ iff*
  $\mathbf{L}_{cd} = \bigcup_{X \in \mathbf{A}(C)} (\Pi_{c.X} \cap \mathbf{A}(d)) \;\cup\; \bigcup_{X \in \mathbf{A}(d)} (\Pi_{d.X} \cap \mathbf{A}(c)) \neq \emptyset.$
  Edge (c,d) is labeled by $\mathbf{L}_{cd}$.

An edge (c,d) of the boundary graph and its label define precisely the attributes that should be eliminated at class level if (c,d) was considered as an instance of a compound. So two pairs of instances (c_1,d_1) and (c_2,d_2) of classes $\mathcal{C}$ and $\mathcal{D}$ should not be considered as instances of the same compound if $\mathbf{L}_{c_1 d_1} \neq \mathbf{L}_{c_2 d_2}$. Fig. 2 illustrates two boundary graphs for which different compound classes will be mined. In this case, we can see that Y is an outer attribute for compound $\{c_1, d_1\}$ while being an inner attribute for compound $\{c_2, d_2\}$. This suggests the following definition:

Definition 4 (dynamic class). *Let $\boldsymbol{\mathcal{BG}}$ be a boundary graph. A dynamic class $\widehat{\mathcal{F}}$ in $\boldsymbol{\mathcal{BG}}$ is a pair $(\mathcal{F}, \mathbf{B})$, where: $\mathcal{F}$ is a compound class; $\mathbf{B} \subseteq \mathbf{A}(\mathcal{F})$ is the set of all the outer attributes of $\mathcal{F}$. Set $\mathbf{B}$ is called $\widehat{\mathcal{F}}$'s boundary.*

Hence, given a dynamic class $\widehat{\mathcal{F}}$, all the nodes in $\mathbf{A}(\widehat{\mathcal{F}})\backslash\mathbf{B}$ are internal and can be eliminated at class level whereas nodes in $\mathbf{B}$ are referenced by other instances and can only be eliminated at instance level. So, to improve structured inference, we shall search the boundary graph for frequent subgraphs, i.e., subgraphs repeated many times, create their corresponding dynamic class, substitute each

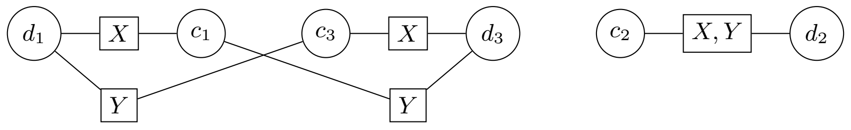


Fig. 2. Different possible connections between instances, resulting in different labels (square nodes). We can see that compound $\{c_1, d_1\}$ is different from compound $\{c_2, d_2\}$.

subgraph by one instance of its dynamic class and, finally, apply an inference algorithm like SVE. However substitutions must be performed carefully: it may actually happen that the occurrences of frequent subgraphs share some nodes. In this case, only one of these occurrences can be substituted else some instances of the "original" system would be counted several times (see (e_1, f_1) and (e_1, f_2) in Fig. 1). Hence the following rule:

Rule 1 *In the boundary graph, substituted subgraphs cannot share any node (i.e. any instance of the original PRM).*

Optimizing structured inference thus amounts to searching for the "best" set of dynamic classes and subgraph substitutions satisfying Rule 1. Unfortunately, as shown in the following proposition, this problem is NP-hard[1]:

Proposition 1. *The following problem is NP-hard:*
Instance: *a PRM, a boundary graph, an integer $K \geq 0$.*
Question: *is there a set of dynamic classes and boundary subgraph substitutions of these classes such that the number of operations (multiplications and summations) performed by structured inference is smaller than K?*

In a sense, this proposition is not very surprising since determining the minimal number of operations in variable elimination algorithms such as SVE or VE is equivalent to determining an optimal elimination sequence, which is known to be NP-hard [22]. In addition, determining all the occurrences of a given subgraph in a graph is NP-hard as well [6]. Finally, given a set of dynamic classes and their subgraph occurrences in the boundary graph, determining which ones should be substituted amounts to solve an *Independent Set* problem in which each vertex represents a boundary subgraph and edges link vertices corresponding to overlapping boundary subgraphs. Again, this problem is NP-hard [6]. However, the proof of Proposition 1 shows that finding the best dynamic classes/substitutions remains NP-hard even in cases where inference in the grounded BN is polynomial (singly-connected BNs). We shall however present in the next subsection an efficient approximate algorithm for determining an effective set of dynamic classes.

4.2 An Approximate Algorithm

The problem of finding frequent patterns in labeled graphs has received many contributions in the literature, although the aim is somewhat different in that it consists of finding subgraphs that appear in many graphs of a database of labeled graphs [12, 16, 26]. However, the connection with our problem is sufficiently high that techniques from this domain can be borrowed to solve our problem. In this paper, we suggest to use a variant of gSpan [26].

The idea consists of creating a search tree $\mathbb{T}$ as follows: each node $N(\widehat{\mathcal{D}})$ of the tree represents a pair $(\widehat{\mathcal{D}}, \mathbb{O}(\widehat{\mathcal{D}}))$ where $\widehat{\mathcal{D}}$ is a dynamic class and $\mathbb{O}(\widehat{\mathcal{D}})$

[1] Proof can be found at http://agrum.lip6.fr/doku.php?id=sum2011

is the set of its instances in the boundary graph $\boldsymbol{\mathcal{BG}}$. In other words, $\mathbb{O}(\widehat{\mathcal{D}})$ is the set of subgraphs of the $\boldsymbol{\mathcal{BG}}$ that fit $\widehat{\mathcal{D}}$. Tree $\mathbb{T}$ is initialized with all the dynamic classes corresponding to 1-edge subgraphs of $\boldsymbol{\mathcal{BG}}$. In $\mathbb{T}$, nodes at level $k+1$ are derived from those at level k by extending their associated subgraph in $\boldsymbol{\mathcal{BG}}$ with one of their adjacent node in $\boldsymbol{\mathcal{BG}}$. As a consequence, each node of $\mathbb{T}$ represents a dynamic class whose boundary subgraph is connected and whose set of instances is nonempty. The whole tree thus reveals precisely all the possible substitutions that can be applied to the PRM. More precisely, $\mathbf{V} = \cup_{N(\widehat{\mathcal{D}})\in\mathbb{T}}\mathbb{O}(\widehat{\mathcal{D}})$ represents the set of substitutions. There just remains to select among $\mathbf{V}$ the "*best*" substitutions possible. To do so, we must enforce Rule 1. It is easily done by observing that each node of $\boldsymbol{\mathcal{BG}}$ can only belong to one dynamic class and, more precisely, to one instance of this dynamic class. Hence, if we create a graph $G = (V, E)$ in which each node of V represents a given element of $\mathbf{V}$, i.e., a subgraph of $\boldsymbol{\mathcal{BG}}$, and each edge $(v_1, v_2) \in E$ represents the fact v_1 and v_2 have a nonempty intersection in $\boldsymbol{\mathcal{BG}}$, then any subset $W \subseteq V$ such that no pair of nodes of W are adjacent in G corresponds to a set of substitutions satisfying Rule 1. In other words, there is a one-to-one mapping between the *Independent Sets* of G and the sets of substitutions satisfying Rule 1. Of course, some substitutions are better than others because they induce higher speed-ups in Structured Inference (see the $\beta_{\widehat{\mathcal{D}}}/\gamma_{\widehat{\mathcal{D}}}$ ratio below). So by weighting nodes of V according to the speed-up improvements they induce, the "*best*" substitutions we look for correspond to solutions of a *Max Weighted Independent Set* problem [9].

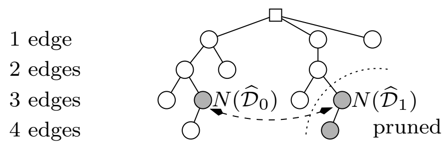


Fig. 3. Dynamic class search tree $\mathbb{T}$.

Of course, the size of $\mathbb{T}$ is exponential and, thus, some pruning is necessary. Pruning rules will be described in the next subsection. But, to guaranty their efficiency, we shall construct $\mathbb{T}$ in such a way that the "best" dynamic classes are constructed first. For this purpose, gSpan defines a linear order that ensures that the more promising the node the smaller its index in the order and suggests to sort all the nodes of each level of $\mathbb{T}$ according to this order [26]. Thus, parsing $\mathbb{T}$ in a depth-first search (DFS) manner guarantees that the "most promising" dynamic classes are constructed first. This leads to the following algorithm:

4.3 Pruning Rules

For the first pruning rule, note that the descendants of a node in $\mathbb{T}$ define the possible extensions of its corresponding dynamic class. Hence, if another node of the search tree corresponds to the same dynamic class (say, e.g., that $\widehat{\mathcal{D}}_1$ is

```
**Input**: A PRM and its boundary graph $\boldsymbol{\mathcal{BG}}$
**Output**: A set of dynamic classes/substitutions
$\mathbb{T} \leftarrow$ all dynamic classes of 1-edge subgraphs of $\boldsymbol{\mathcal{BG}}$
sort the nodes in $\mathbb{T}$ according to the gSpan linear order
parse $\mathbb{T}$ in a DFS manner
**foreach** *node* $N(\widehat{\mathcal{D}})$ *visited* **do**
    create the children of $N(\widehat{\mathcal{D}})$, sort them w.r.t. gSpan's linear order and add
    them to $\mathbb{T}$
    prune the "unpromising" children
**end**
solve a Max Weighted Independent Set
**return** *the set of "best" dynamic classes/substitutions*
```

Algorithm 1: Computing dynamic classes/substitutions.

the same class as $\widehat{\mathcal{D}}_0$), then both nodes and their descendants represent identical dynamic classes. So, $N(\widehat{\mathcal{D}}_1)$ and its descendants can be safely pruned from the search. Determining whether two nodes represent the same dynamic class is simply achieved through gSpan's canonical labeling of subgraphs (see [26] for a detailed description).

The second rule is related to the gain achievable in Structured Inference using dynamic classes: nodes $N(\widehat{\mathcal{D}})$ that define classes whose subgraph substitutions do not speed-up Structured Inference can be pruned. To estimate the gain in speed, recall that, by Rule 1, only a subset of the subgraphs of $\mathbb{O}(\widehat{\mathcal{D}})$ can be substituted in $\boldsymbol{\mathcal{BG}}$ by instances of $\widehat{\mathcal{D}}$. Let $s_{\widehat{\mathcal{D}}}$ denote the cardinal of this subset. The number of operations (multiplications, additions) performed by Structured Inference on these substitutions is equal to $w_{\widehat{\mathcal{D}}} + s_{\widehat{\mathcal{D}}} \times \overline{w}_{\widehat{\mathcal{D}}}$, where $w_{\widehat{\mathcal{D}}}$ and $\overline{w}_{\widehat{\mathcal{D}}}$ denote the number of operations necessary to eliminate $\widehat{\mathcal{D}}$'s inner nodes at class level and $\widehat{\mathcal{D}}$'s outer nodes at instance level respectively. Now remember that, in tree $\mathbb{T}$, $\widehat{\mathcal{D}}$ corresponds to a 1-edge extension of its parent $\pi(\widehat{\mathcal{D}})$. So, the subgraphs of $\mathbb{O}(\widehat{\mathcal{D}})$ that were not substituted are 1-edge extensions of subgraphs of $\mathbb{O}(\pi(\widehat{\mathcal{D}}))$. Assuming that they were all substituted as instances of $\pi(\widehat{\mathcal{D}})$, their eliminations by Structured Inference would have cost $w_{\pi(\widehat{\mathcal{D}})} + (|\mathbb{O}(\widehat{\mathcal{D}})| - s_{\widehat{\mathcal{D}}}) \times \overline{\overline{w}}_{\widehat{\mathcal{D}}}$ where $\overline{\overline{w}}_{\widehat{\mathcal{D}}} = \overline{w}_{\pi(\widehat{\mathcal{D}})} + k_{\widehat{\mathcal{D}}}$ and $k_{\widehat{\mathcal{D}}}$ corresponds to the elimination of the edge added to $\pi(\widehat{\mathcal{D}})$. So the total cost incurred by the exploitation of $N(\widehat{\mathcal{D}})$ is $\beta_{\widehat{\mathcal{D}}} = w_{\widehat{\mathcal{D}}} + w_{\pi(\widehat{\mathcal{D}})} + s_{\widehat{\mathcal{D}}} \times \overline{w}_{\widehat{\mathcal{D}}} + (|\mathbb{O}(\widehat{\mathcal{D}})| - s_{\widehat{\mathcal{D}}}) \times \overline{\overline{w}}_{\widehat{\mathcal{D}}}$ whereas, by just exploiting $\pi(\widehat{\mathcal{D}})$, it would have been $\gamma_{\widehat{\mathcal{D}}} = w_{\pi(\widehat{\mathcal{D}})} + |\mathbb{O}(\widehat{\mathcal{D}})| \times \overline{\overline{w}}_{\widehat{\mathcal{D}}}$. So, class $\widehat{\mathcal{D}}$ is unattractive for inference and $N(\widehat{\mathcal{D}})$ may be pruned whenever $\alpha_{\widehat{\mathcal{D}}} = \beta_{\widehat{\mathcal{D}}} - \gamma_{\widehat{\mathcal{D}}} = w_{\widehat{\mathcal{D}}} + s_{\widehat{\mathcal{D}}} \times (\overline{w}_{\widehat{\mathcal{D}}} - \overline{\overline{w}}_{\widehat{\mathcal{D}}}) > 0$. Finally, note that $s_{\widehat{\mathcal{D}}}, w_{\widehat{\mathcal{D}}}, \overline{\overline{w}}_{\widehat{\mathcal{D}}}, k_{\widehat{\mathcal{D}}}$ can be estimated quickly: as shown in the preceding subsection, $s_{\widehat{\mathcal{D}}}$ can be estimated by solving a *Max Independent Set* problem induced by $\mathbb{O}(\widehat{\mathcal{D}})$. To estimate $w_{\widehat{\mathcal{D}}}$, it is sufficient to compute a junction tree of $\widehat{\mathcal{D}}$'s DAG [23], eliminating only inner nodes, and to sum-up the sizes of its cliques. Eliminating the remaining variables provides an estimation of $\overline{w}_{\widehat{\mathcal{D}}}$. $k_{\widehat{\mathcal{D}}}$ can be estimated similarly.

Note however that $\mathbb{T}$ is not α-decreasing, i.e., it may happen that $\alpha_{\widehat{\mathcal{D}}} > 0$ for a given node $N(\widehat{\mathcal{D}})$, but not for some of its descendants. This property results from the fact that, in these descendants the number of inner nodes may be far higher than that in $\widehat{\mathcal{D}}$, hence decreasing $w_{\widehat{\mathcal{D}}}$ (dropping constraints on the junction tree's elimination order) as well as $\overline{w}_{\widehat{\mathcal{D}}}$ (the inner nodes do not belong to the boundary). The α-non-decreasing property does not allow for a clear pruning rule. In the paper, we used the following rule: whenever a node in $\mathbb{T}$ had an $\alpha_{\widehat{\mathcal{D}}} > 0$, we pruned the node and its descendants.

5 Experimental Results

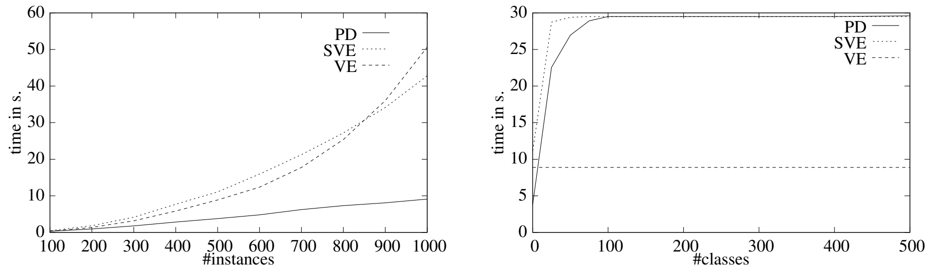
We now describe different set of experiments that highlight the gain in inference speed resulting from the combination of structured inference and pattern mining. In each experiment, we compared our new algorithm (subsequently denoted as PD for Pattern Discovery) with Structured Variable Elimination (SVE), the standard inference algorithm for structured inference [24], and also with Variable Elimination (VE), a classic and standard probabilistic inference algorithm for Bayesian Networks [4]. Response times reported for PD take into account both pattern mining and inference. For experiments using VE, results include both grounding and inference time. It is important to note that our experiments included no evidence. This choice was motivated by the fact that the structure of the network varies drastically given evidence. Our goal here was to show how pattern mining can improve inference when there exist repetitions in the network. Moreover, evidence is not a good indicator of repetitions as it can either be identically applied in each pattern, thus preserving repetition, or applied randomly, thus breaking the structure. Experiences 1 and 2 show the results of our new approach on networks with and without repetitions, hence providing a good insight of PD's performance. All our experiments were performed on an Intel Xeon at 2.7 Ghz. The source code of our PRMs implementation, the inference algorithm and the generation algorithms can be found in the aGrUM project[2].

The key to understand these experimentations lies in the generation of the benchmarked PRMs. High level frameworks such as PRMs offer a wide variety of generation methods. Here, our primary concern was the generation of PRMs in which we could control the amount of structure repetition in order to prove that, when confronted to a large amount of pattern repetitions, i) a substantial speed gain can be achieved and ii) our approach does not suffer from a prohibitive pattern mining cost. Our generator takes the following parameters as inputs: *domain* is the domain size of each attribute; min_{attr} is the number of attributes common to all classes; max_{attr} is the number of attributes in each class; c is the minimal number of classes; max_{ref} is the maximal number of reference slots allowed per class; n is the number of instances in the system.

The PRM's generation process is performed as follows: first, we generate an interface[3] with min_{attr} attributes which will be implemented by all classes

[2] http://agrum.lip6.fr

[3] If a class implements a given interface, then it guarantees the existence of the attributes and reference slots defined in that interface [25].



(a) First set of experiments. (b) Second set of experiments.

Fig. 4. Structural repetition is an important factor for PD's performance. Unsurprisingly, performance decrease dramatically for systems with no structural repetition.

and will be the slot type of each reference slot in each class. Next, for all $k \in [0, \ldots, \max_{ref}]$, a class with precisely k reference slots is created. Then, if $max_{ref} < c$, we generate new classes until exactly c classes have been created. For those new classes, the number of reference slots is chosen randomly between 0 and max_{ref}. Finally, we generate a DAG S representing the relational skeleton of our generated system: each node represents an instance and an arc $i \to j$ represents the fact that there exists $\rho \in \mathbf{R}(j)$ such that $i = j.\rho$. For a given node i with π_i parents in S, we instantiate a class randomly chosen among all the classes with precisely π_i reference slots. A given class C is generated as follows: we first create a DAG G_C with max_{attr} nodes, we then add to C k reference slots and max_{attr} attributes. Dependencies between attributes are defined using G_C. For each reference slot ρ, we create a slot chain $\rho.A$, where $A \in \mathbf{A}(\mathcal{I})$ is chosen randomly among all the attributes in $\mathbf{A}(\mathcal{I})$. The slot chain is then added as a parent of an attribute of C chosen randomly. DAGs are generated using the algorithm provided in [11].

In our first set of experiments, we generated systems with an increasing number of instances. Each class contains 15 attributes ($max_{attr} = 15$), each attribute's domain size is equal to 4 ($domain = 4$) and each class has at most 4 incoming arcs ($max_{ref} = 4$). Finally, the minimal amount of classes required was set to $c = 5$, which implies that there are precisely $max_{ref} + 1 = 5$ classes in each system. These experiments highlight the behavior of PD when many repetitions can be found in the system. Fig. 4(a) shows the response times of PD, SVE and VE when no evidence is observed and with a number of instances varying from 100 to 1000. Clearly, in this case, PD significantly outperforms both VE and SVE.

An important factor is the ratio of PD's inference time over that of SVE. The gain of PD against SVE and that of SVE against VE are due to the presence of structural repetition in the generated networks. It can be seen that SVE's complexity is less impacted by the size of the system than VE's complexity. But for small systems with small classes, SVE does not guarantee a considerable speed gain. By exploiting pattern mining, PD significantly increases the gain

Table 1. Third experiment: patterns mining efficiency for PD. Values are averages. Inst. stands for instances, attr. for attributes and pat. for pattern.

#inst.	#pat.	pat.	max pat.	#inst.	max inst.	% of attr.
		repetition	repetition	per pat.	per pat.	in a pat.
200	11.88	2.92	6.26	2.15	4.08	37.29%
400	24.68	3.40	10.46	2.25	4.71	47.20%
600	36.35	3.91	15.92	2.36	5.25	55.90%
800	46.51	4.50	20.25	2.45	5.62	64.09%
1000	54.19	5.25	30.07	2.62	6.12	75.54%

obtained by repetition. Thus, where SVE does not perform well compared to VE, PD infers larger patterns that can drastically increase performance. In our first experiments, there is enough structure to see the possible gain provided by our new approach. Yet, we must also consider cases where there are few or even no structural repetitions. The amount of pattern repetitions can be influenced by the number of classes, so if we increase that number we should observe a less favorable ratio between PD's and SVE's inference time against VE. This is the purpose of our second set of experiments.

In our second set of experiments we generated systems with an increasing number of classes ($c \in [0, 500]$) and 500 instances. The remaining parameters are equal to those of the first experiment. The goal here is twofold: we want to show that, when no structure is exploitable, there is no overhead in proceeding with the pattern mining and that pattern repetitions is critical for PD's performance. Fig. 4(b) shows that when the number of classes increases dramatically, the speed gain induced by PD and SVE are considerably less significant. If we compare those results with those obtained by VE, we see that PD and SVE are considerably counter-performing. To anyone familiar with structural inference, this is an unsurprising result and these results can be explained by the fact that the elimination order used by PD and SVE (inner attributes before outer attributes) is in most cases suboptimal. If PD and SVE show better results than VE in Fig. 4(a) it is only because the gain resulting from the reduction of redundant computations compensate the suboptimal elimination order. Fortunately, detecting repetition is trivial in an object-oriented framework as the amount of instantiations of each class is a good indicator of structural repetition. The presence of evidence is also a good indicator, as different evidence will break down the structure and thus reduce the amount of repetition in the network. We can easily switch to classic inference if needed by detecting situations which would lead to counter-performing results: few instantiations of each classes, heavy evidence, seemingly random evidence. Finally, we observe no over-cost due to pattern mining. This is also an unsurprising result as our pruning rules take into account frequencies and cut the mining process when such value is too low (here the minimal frequency allowed was set to 2).

In our third experiment, we analyze the amount of patterns found by PD with the parameters from experiment 1 ($max_{attr} = 15, domain = 4, max_{ref} = 4, c = 5, max_{ref} + 1 = 5$). The results of this experiment are summarized in Tab. 1.

A noticeable point is the low number of instances in each pattern. This is a consequence of our pruning rule which was designed to be strict. It favors smaller patterns because larger ones are in most cases less cost effective (they often induce a larger clique than an optimal elimination order would) and because they are less frequent. In general, discovered patterns consisted of few small patterns largely repeated and many different patterns less repeated. The latter were used to fill-in the gaps in the structure once the main patterns were applied. If we consider the last column of Tab. 1 we can see that the larger a system, the more the attributes covered. The fact that the coverage increases with the system size explains why the inference time of PD increases linearly with the system size: the large number of usable patterns compensates the complexity induced by the number of instances.

To conclude our experiments, we applied PD to a classic BN: the Pigs network. This network is remarkable in that it only contains two distinct CPTs, which are represented in our framework by two classes. The network in itself is too small to point out any significant gain in inference time, however it is still interesting to analyze the patterns found by PD. Our approach mined 14 different patterns. On average, they are repeated 11 times and the maximal amount of repetitions equals 45. Only patterns with 2 instances are found. Discovered patterns cover up to 69% of the 441 attributes present in the Pigs network. As for our previous results, our pruning rules favor smaller patterns since larger ones tend to be less cost effective and less frequent. While the size of the Pigs network does not enable to point out the efficiency of our approach in terms of inference time, the existence of such structures and the results we obtained over random networks can help conclude to the efficiency of our approach. We can also point out its usefulness w.r.t. modeling: by pointing out frequent patterns in a system we can infer new classes which can then be used by experts for modeling purposes.

6 Conclusion

In this paper, we showed that mining patterns can significantly alleviate inference costs. Although finding the optimal set of patterns is NP-hard, we provided an efficient approximate mining algorithm. Our experimental results confirm that this approach can lead to a significant improvement of inference tasks in PRM. But there is still room for improving inference in PRMs. For instance, our approach, especially its pruning, can still be improved. In addition, many refinements of the PRM framework like class inheritance, structural uncertainty or multiple references, should be used to speed-up inference.

References

1. Bangsø, O.: Object Oriented Bayesian Networks. Ph.D. thesis, Aalborg University (March 2004)

2. Bangsø, O., Sønderberg-Madsen, N., Jensen, F.: A bn framework for the construction of virtual agents with human-like behaviour. In: Proc. of PGM'06 (2006)
3. Bangsø, O., Wuillemin, P.H.: Top-down construction and repetitive structures representation in Bayesian networks. In: Proc. of FLAIRS'00. pp. 282–286 (2000)
4. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence 113, 41–85 (1999)
5. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proc. of IJCAI'99 (1999)
6. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
7. Getoor, L., Friedman, N., Koller, D., Pfeffer, A., Taskar, B.: Probabilistic relational models. In: Getoor, L., Taskar, B. (eds.) An Introduction to Statistical Relational Learning, chap. 5. MIT Press (2007)
8. Getoor, L., Koller, D., Taskar, B., Friedman, N.: Learning probabilistic relational models with structural uncertainty. In: ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries (2000)
9. Halldórsson, M.: Approximations of weighted independent set and hereditary subset problems. Journal of Graph Algorithms and Applications (2000)
10. Heckerman, D., Meek, C., Koller, D.: Probabilistic models for relational data. Tech. rep., WA: Microsoft Corporation, Redmond (2004)
11. Ide, J.S., Cozman, F.G., Ramos, F.T.: Generating random Bayesian networks with constraints on induced width. In: Proc. of ECAI'04. pp. 323–327 (2004)
12. Inokuchi, A., Washio, T., Motoda, H.: A general framework for mining frequent subgraphs from labeled graphs. Fundamenta Informaticae (2005)
13. Jaeger, M.: Relational Bayesian networks. In: Proc. of UAI'97 (1997)
14. Kersting, K., Raedt., L.D.: Bayesian logic programs. Technical report no. 151, Institute for Computer Science, University of Freiburg, Germany (April 2001)
15. Koller, D., Pfeffer, A.: Object-oriented Bayesian networks. In: Proc. of AAAI'97. pp. 302–313 (1997)
16. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proc. of ICDM'01 (2001)
17. Madsen, A., Jensen, F.: LAZY propagation: A junction tree inference algorithm based on lazy inference. Artificial Intelligence 113(1–2), 203–245 (1999)
18. Mahoney, S., Laskey, K.: Network engineering for complex belief networks. In: Proc. of UAI'96 (1996)
19. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman (1988)
20. Pfeffer, A., Koller, D., Milch, B., Takusagawa, K.: SPOOK: A system for probabilistic object-oriented knowledge representation. In: Proc. of UAI'99 (1999)
21. Pfeffer, A.: Probabilistic Reasoning for Complex Systems. Ph.D. thesis, Stanford University (2000)
22. Rose, D., Lueker, G., Tarjan, R.: Algorithmic aspects of vertex elimination on graphs. SIAM J. on Computing 5, 266–283 (1976)
23. Rose, D.: Triangulated graphs and the elimination process. J. Math. Analysis and Applications (1970)
24. Torti, L., Wuillemin, P.H.: Structured value elimination with d-separation analysis. In: Proc. of FLAIRS'10. pp. 122–127 (2010)
25. Torti, L., Wuillemin, P.H., Gonzales, C.: Reinforcing the object-oriented aspect of probabilistic relational models. In: Proc. of PGM'10 (2010)
26. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proc. of ICDM'02 (2002)