Speeding-up Structured Probabilistic Inference using Pattern Mining

Lionel Torti, Christophe Gonzales, Pierre-Henri Wuillemin

Laboratoire d'Informatique de Paris 6 (LIP6/UPMC)
4 place Jussieu, 75005 Paris, FRANCE

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 such as Probabilistic Relational Models (PRM) thus offer an effective way to represent such systems. They 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, object oriented-based framework's 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 online 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.

Keywords: Probabilistic Relational Models; Bayesian networks; inference; pattern mining.

1. Introduction

Bayesian networks (BN) [1] 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 [2, 3]. 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, the BN's design is static: any change in the topology of their graphical structure induces significant costs to update their sets of conditional probability tables.

Solving these problems has been the main concern of several BN extensions using the object-oriented paradigm [4, 2, 5, 6], including Probabilistic Relational Models (PRM). Besides, first-order logic extensions were proposed to offer more expressive power than the propositional framework offered by BNs [7, 8]. Learning being a critical problem when exploiting BNs over large knowledge bases, entity-relationship extensions were also proposed for relational learning [9, 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 Models community has worked actively on FOPMs and object-oriented models have been somewhat neglected: since the introduction of Object-Oriented Bayesian Networks [11, 4], the amount of contributions on object-oriented probabilistic graphical models has actually been relatively small [12, 13, 9]. 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. Indeed, in the context of an applied research, namely the SKOOB ANR project [14], we observed that the modeling of industrial processes, biological behavior, physical or chemical systems, etc., implies a methodological approach based on the design of components (e.g., numerous identical valves, many assembly lines, many cellular structures, repetition of elementary cells, etc.). It also often involves top-down design where the components are integrated with other components in a hierarchical construction.

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 FOPMs. For further details on the comparison between FOPMs, OOBNs and other component-based approaches like Multi-Sectioned Bayesian Network (MSBN, [15]), we refer the readers to [6].

Object-oriented frameworks and, *a fortiori*, PRMs 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 the definition of a class 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 once within classes, caching them and using the cache for all their instances. This process allows Structured Inference algorithms like *Structured Variable Elimination* (SVE) to outperform classical BN inference engines by orders of magnitude [5].

It is important to emphasize the differences between structured inference and lifted inference. The latter is a probabilistic inference scheme that exploits FOPMs to *lift* identical worlds and reduce the amount of computations w.r.t. a given query and evidence [16, 17, 18, 19, 20]. In this paper, our use of PRMs differs from their original use in [5] as we exploit them to model closed world systems, i.e., systems with no structural uncertainty (see [21] for more details). In such cases, lifted inference does not offer any substantial gain compared to a classical BN inference as there is nothing to *lift*. Consequently, we cannot compare both approaches as they do not apply to similar systems. Structured inference and lifted inference are therefore not rival inference schemes but rather complementary optimization techniques.

In real world applications, instances are often combined and form patterns repeated many times throughout the network. Unfortunately, to keep construction and maintenance costs at a low level, it is often the case that such patterns are not encoded as classes by experts. For instance, in genetics, classes usually encode relationships between a child and its parents, but they seldom encode those among whole families (children, parents, grandparents, etc). Therefore, if a problem concerns many families with two children, this pattern should be exploited to speed-up inference.

In this paper, we thus propose an enhancement of structured inference for Probabilistic Relational Models [22, 23] that addresses this problem. By using a frequent subgraph pattern mining algorithm, it is possible to discover such repeated combinations, and their exploitation can actually speed-up significantly structured inference. Mining optimally such patterns is time expensive and, actually, we prove that it is a NP-hard problem. However, we provide an efficient approximate algorithm that can quickly find patterns that improve structured inference response times. This is confirmed by our experiments.

The paper is organized as follows: Section 2 recalls the basics of object-oriented frameworks using PRMs. Section 3 describes structured inference. Then, in Section 4, the pattern discovery problem is specified and it is proved that its optimal resolution is NP-hard. Section 5 provides our approximate algorithm. Experiments reported in Section 6 show the practical efficiency of our approach. Finally, concluding remarks are given in Section 7. All proofs are given in an appendix in Section 8.

2. Description of Probabilistic Relational Models (PRM)

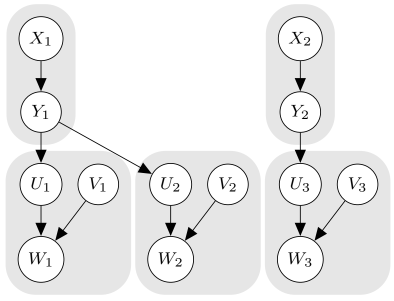
PRMs were first proposed for relational learning [24]. By their definition, they can be viewed as an extension of Object Oriented Bayesian Networks [5] and, actually, they offer a sound object-oriented framework [23]. We shall first introduce them in Subsection 2.1 as presented in [23]. Then, in Subsection 2.2, we will extend PRMs adding the notion of an interface. For an intuitive presentation of PRM's key concepts, the reader should refer to [25].

2.1. Basics of PRMs

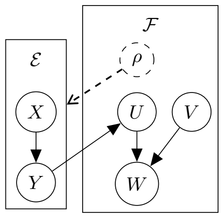
In order to properly define PRMs, we shall use the following cross-definitions[1]:

Definition 1 (Attribute, Reference slot, Class).

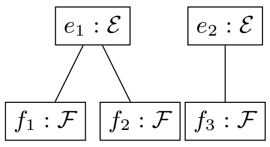
- *An* attribute $\mathcal{C}.X$ *is a random variable enclosed in a class* $\mathcal{C}$*.* $\mathcal{C}$ *is called the resident class of* X*.*
- *A* reference slot $\mathcal{C}.\rho$ *is a local name enclosed in class* $\mathcal{C}$ *for another class, say* $\mathcal{D}$*. Then, class* $\mathcal{D}$ *grants the access to all of its attributes and reference slots to* $\mathcal{C}$*.* $\text{Range}(\rho)$ *denotes class* $\mathcal{D}$*.* $\text{Domain}(\rho)$ *denotes the resident class of* ρ *(i.e.* $\mathcal{C}$*).*
- *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,*
  - $\mathbf{R}(\mathcal{C})$ *is a set of reference slots. The set* $\overline{\mathbf{A}(\mathcal{C})}$ *of all the attributes that are located within* $\mathcal{C}$*, i.e.,* $\mathbf{A}(\mathcal{C})$*, or that are reachable by way of reference slots 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 conditional probabilistic tables (CPTs) of all attributes* $X \in \mathbf{A}(\mathcal{C})$ *conditionally to their parents* Π_X *in* $G(\mathcal{C})$*.*



(a) A BN.



(b) Classes $\mathcal{E}$ and $\mathcal{F}$.



(c) The BN as a system.

$$P(Y_1|X_1) = \begin{array}{c|c|c|} & x_1 & \overline{x_1} \\ \hline y_1 & a & b \\ \hline \overline{y_1} & c & d \\ \hline \end{array} \qquad P(Y_2|X_2) = \begin{array}{c|c|c|} & x_2 & \overline{x_2} \\ \hline y_2 & a & b \\ \hline \overline{y_2} & c & d \\ \hline \end{array}$$

$$P(U_1|Y_1) = \begin{array}{c|c|c|} & y_1 & \overline{y_1} \\ \hline u_1 & e & f \\ \hline \overline{u_1} & g & h \\ \hline \end{array} \qquad P(U_2|Y_1) = \begin{array}{c|c|c|} & y_1 & \overline{y_1} \\ \hline u_2 & e & f \\ \hline \overline{u_2} & g & h \\ \hline \end{array}$$

$$P(U_3|Y_2) = \begin{array}{c|c|c|} & y_2 & \overline{y_2} \\ \hline u_3 & e & f \\ \hline \overline{u_3} & g & h \\ \hline \end{array}$$

(d) The CPTs of the BN.

$$P(Y|X) = \begin{array}{c|c|c|} & x & \overline{x} \\ \hline y & a & b \\ \hline \overline{y} & c & d \\ \hline \end{array}$$

$$P(U|\mathcal{E}.Y) = \begin{array}{c|c|c|} & y & \overline{y} \\ \hline u & e & f \\ \hline \overline{u} & g & h \\ \hline \end{array}$$

(e) The class CPTs.

Figure 1: Analysis of a fragment of BN (a) and (d) reveals the use of two recurrent patterns, which are confined into two classes (b) whose CPTs are defined in (e). Hence, a system equivalent to figure (a) may be built (c). The CPTs of the BN and of the classes are defined in (d) and (e) respectively, where elements a, b, c, d, e, f, g, h represent probabilities and BN random variables are all assumed to be Boolean.

Fig. 1.(b) shows a set of classes enabling to encode the BN fragment of Fig. 1.(a). Class $\mathcal{E}$ is defined as $\langle \mathbf{A}(\mathcal{E}), \mathbf{R}(\mathcal{E}), G(\mathcal{E}), \mathbf{P}(\mathcal{E})\rangle$, where $\mathbf{A}(\mathcal{E}) = \{X, Y\}$ is the set of attributes encapsulated into $\mathcal{E}$, $\mathbf{R}(\mathcal{E}) = \emptyset$ since no node in class $\mathcal{E}$ is the child of another node that does not belong to $\mathcal{E}$. $G(\mathcal{E})$ is the graph whose nodes are $\{X, Y\}$ and whose arcs are $\{(X, Y)\}$. Finally, $\mathbf{P}(\mathcal{E}) = \{P(X), P(Y|X)\}$. Note that the same table $P(Y|X)$ defined on Fig. 1.(e) is used as the CPT for all nodes Y_j, $j = 1, 2$, in the BN fragment (see Fig. 1.(a) and 1.(d)). As such, this mechanism enables to compactly define the whole BN. Similarly, class $\mathcal{F}$ is defined as $\langle \mathbf{A}(\mathcal{F}), \mathbf{R}(\mathcal{F}), G(\mathcal{F}), \mathbf{P}(\mathcal{F})\rangle$,

[1]Note that we use standard Object Oriented notation "." to indicate encapsulation.

where $\mathbf{A}(\mathcal{F}) = \{U, V, W\}$, $\mathbf{R}(\mathcal{F}) = \{\mathcal{F}.\rho\}$ with $\mathcal{F}.\rho$ the reference slot pointing toward $\mathcal{E}$, i.e., $\textsc{Range}(\mathcal{F}.\rho) = \mathcal{E}$. Graph $G(\mathcal{F}) = (\{\mathcal{E}.Y, U, V, W\}, \{(\mathcal{E}.Y, U), (U, W), (V, W)\})$ and the set of conditional probability tables of $\mathcal{F}$ is $\mathbf{P}(\mathcal{F}) = \{P(U|\mathcal{E}.Y), P(V), P(W|U, V)\}$. Again, table $P(U|\mathcal{E}.Y)$ of Fig. 1.(e) defines completely the CPTs of nodes U_j, $j = 1, 2, 3$, in the BN fragment.

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 a 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 $\mathcal{B}$ such that:*
  - *there exists a one-to-one mapping between c and $\mathbf{A}(\mathcal{C})$. We note $c.X$ the random variable in c corresponding to the attribute $\mathcal{C}.X$,*
  - *for each reference slot $\mathcal{C}.\rho$ of $\mathcal{C}$, there exists an instance r of class $\textsc{Range}(\mathcal{C}.\rho)$ such that if $\mathcal{C}.X$ has a parent $\textsc{Range}(\mathcal{C}.\rho).Y$ then $c.X$ has a parent $r.Y$ in $\mathcal{B}$.*
    We then note $\textsc{Range}(c.\rho) = r$ and $\textsc{Domain}(c.\rho) = c$.
  - *The CPT of $c.X$ in $\mathcal{B}$ is the same as that of $\mathcal{C}.X$ in class $\mathcal{C}$.*

  By abuse of notation, every notion and notation defined for a class also applies to instances ($\mathbf{A}(c)$, $\mathbf{R}(c)$, $\textsc{Range}$, $\textsc{Domain}$, resident instance, etc.).
- *A system S is the representation of $\mathcal{B}$ in the PRM framework: it is a finite set of instances such that:*
  - *S forms a partition of $\mathcal{B}$,*
  - *$\forall i \in S, \forall \rho \in \mathbf{R}(i), \exists j \in S$ such that $\textsc{Range}(i.\rho) = j$.*

Fig. 1.(c) illustrates the notion of a system: here, a system is simply the set of instances $\{e_1, e_2, f_1, f_2, f_3\}$. Note that all reference slots are actually bound. For instance, $\textsc{Range}(f_1.\rho) = e_1$. Definition 2 enforces the "closed world" feature of systems, i.e., they are finite sets of instances with all reference slots properly bound. As mentioned in the introduction, this constraint is reasonable for complex systems of many domains. For instance, to reason on industrial milk fermenters, all pipe connections need to be fully specified.

Using a graph whose nodes are instances instead of random variables allows to have a very synthetic view of a complex BN. For instance, Fig. 1.(c) is much more compact than Fig. 1.(a). This graphical representation is called a *relational skeleton*.

Definition 3 (Relational skeleton, ground BN).

- *The graph representing instances by nodes and connections between range and resident instances by edges is called the relational skeleton of S.*
- *The Bayesian network corresponding to a relational skeleton is called the* ground BN *or the* grounded BN*.*

Note that the BN fragment of Fig. 1.(a) has two connected components, i.e., the random variables on its left part are independent of those on its right part. In terms of relational skeletons, this corresponds to the latter having several connected components as well (as shown in Fig. 1.(c)). Of course, a BN with only one connected component, which is most common in practice, can also be described in terms of PRMs and, in this case, the corresponding relational skeleton also has only one connected component.

We now extend this basic PRM framework by importing into it the object oriented notion of an interface. This results in a new probabilistic language with an increased expressive power.

2.2. Interface

Inheritance in PRMs, and more generally in Object Oriented frameworks, is a complex feature. A complete description of the inheritance paradigm proposed in our PRM framework would be out of the scope of this paper and the interested readers shall refer to [23, 6] for details. However, the notion of an interface is closely related to that of inheritance and we will need it in the next sections. So, we will now briefly define a "light" notion of interface.

When some attribute, say $\mathcal{D}.Y$, of a class $\mathcal{D}$ is addressed within some class $\mathcal{C}$ through one of its reference slot $\mathcal{C}.\rho$, this implies that $\mathcal{D}.Y$ is a parent of some attribute of $\mathcal{C}$, say $\mathcal{C}.X \in \mathbf{A}(\mathcal{C})$. Consequently, $\mathcal{D}.Y$ shall be one the variables on the right hand side of the conditioning bar of the CPT assigned to $\mathcal{C}.X$. Now, for this CPT to be correct, it is not absolutely necessary that Y belongs precisely to class $\mathcal{D}$. Rather, the weaker condition that Y has the same domain as $\mathcal{D}.Y$ is sufficient to ensure that the CPT of $\mathcal{C}.X$ is valid. Hence, we can relax the constraint that the range of a reference slot is necessarily a given class and only require that the attributes referenced have a specified domain. We shall see below that this relaxation is necessary, for instance, to model dynamic Bayesian networks (dBN) by PRMs. This suggests the notion of interface below, which represents exactly the minimum information needed for a reference slot to be used as a container of parents (in order to have a valid joint probability distribution).

Definition 4 (Interface, implementation).

- *An interface $\mathcal{I}$ is defined by a set of attributes, denoted $\mathbf{A}(\mathcal{I})$, and by a set of reference slots, denoted $\mathbf{R}(\mathcal{I})$. Interfaces cannot be instantiated.*
- *A class $\mathcal{C}$ implements an interface $\mathcal{I}$ (denoted by $\mathcal{C} \blacktriangleleft \mathcal{I}$) if:*
    - $\mathbf{A}(\mathcal{I}) \subseteq \mathbf{A}(\mathcal{C})$
    - $\mathbf{R}(\mathcal{I}) \subseteq \mathbf{R}(\mathcal{C})$

The only distinction between a class and an interface is that the latter does not contain any CPT. This feature thus prevents it from being instantiated. For a class that implements an interface, the latter is just a contract that enforces the presence of a set of attributes and a set of reference slots inside the class. Note that the class will have to specify CPTs for all the attributes of the interface.

In order to use this notion of interface, the definition of reference slot needs to be slightly modified:

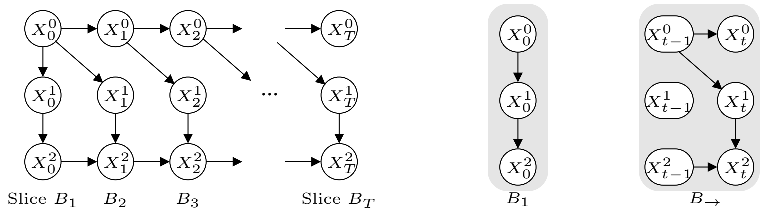
Definition 5 (Reference slot with interface).

A reference slot of a class $\mathcal{C}$ is a local name for a class or an interface giving access to all of its attributes and reference slots from within $\mathcal{C}$.

- *If the* Range *of a reference slot $\mathcal{C}.\rho$ is a class then, for every instance c of $\mathcal{C}$,* $\text{Range}(c.\rho)$ *is an instance of the class* $\text{Range}(\mathcal{C}.\rho)$*.*
- *If the* Range *of a reference slot $\mathcal{C}.\rho$ is an interface then, for every instance c of $\mathcal{C}$,* $\text{Range}(c.\rho)$ *is an instance of a class that implements the interface* $\text{Range}(\mathcal{C}.\rho)$*.*

Interfaces and reference slots over interfaces prove to be useful, for instance, to model dBNs by PRMs: actually, a dBN represents the uncertainties about the state of a system evolving over some discrete time horizon [26] (Figure 2a). A 2TBN is a compact representation for a dBN defined over this horizon using a pair of BNs $(B_1, B_\rightarrow)$. B_1 is the BN fragment used in time slice 1 and $B_\rightarrow$ is used for all the other slices (Figure 2b). Of course, the nodes in the BN fragments of consecutive time slices can be dependent and, therefore, there exist some arcs across consecutive time slices.

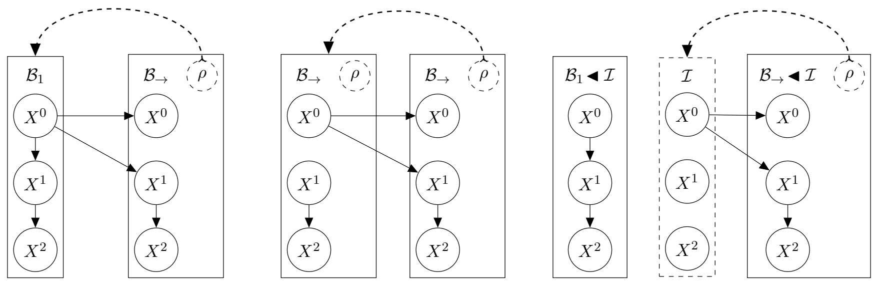
Intuitively, to model such problem using PRMs, one would create a class $\mathcal{B}_1$ for B_1 and a class $\mathcal{B}_\rightarrow$ for $B_\rightarrow$. However, Definition 1 would not allow this. Actually, an arc, say (X_1^i, X_2^j) linking some node in time slice 1 to another node in time slice 2, would require that class $\mathcal{B}_\rightarrow$ contains a reference slot $\mathcal{B}_\rightarrow.\rho$ pointing to class $\mathcal{B}_1$ so that $\text{Range}(\mathcal{B}_\rightarrow.\rho).X^i = \mathcal{B}_1.X^i$ is a parent of $\mathcal{B}_\rightarrow.X^j$ (Figure 3a). But, by definition of 2TBNs, for any time $t > 2$, arc (X_t^i, X_{t+1}^j) shall also belong to the dBN and this would require that reference



(a) A dynamic BN (dBN). (b) The equivalent 2TBN.

Figure 2: A dBN and its compact representation as a 2TBN

slot $\mathcal{B}_\rightarrow.\rho$ points to $\mathcal{B}_\rightarrow$ instead of $\mathcal{B}_1$ so that $\text{Range}(\mathcal{B}_\rightarrow.\rho).X^i = \mathcal{B}_\rightarrow.X^i$ is a parent of $\mathcal{B}_\rightarrow.X^j$ (Figure 3b). Thus, the strict application of Definition 1 would rule out modeling dBNs by PRMs.


$\mathcal{B}_1$
$\mathcal{B}_\rightarrow$
ρ
X^0
X^1
X^2
$\mathcal{B}_1 \blacktriangleleft \mathcal{I}$
$\mathcal{I}$
$\mathcal{B}_\rightarrow \blacktriangleleft \mathcal{I}$

(a) Relations between $\mathcal{B}_1$ and $\mathcal{B}_\rightarrow$ for instances B_1 and B_2 : $\text{Range}(\mathcal{B}_\rightarrow.\rho) = \mathcal{B}_1$

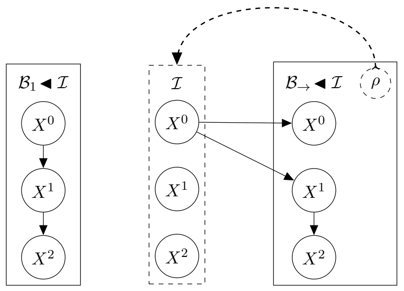
(b) Relations between $\mathcal{B}_\rightarrow$ and itself for instances B_{t-1} and B_t : $\text{Range}(\mathcal{B}_\rightarrow.\rho) = \mathcal{B}_\rightarrow$.

(c) Resolution of the dilemma using an interface implemented by both $\mathcal{B}_1$ and $\mathcal{B}_\rightarrow$: $\text{Range}(\mathcal{B}_\rightarrow.\rho) = \mathcal{I}$.

Figure 3: The dilemma of modeling a dBN as a PRM.

Now, create an interface $\mathcal{I}$ without any reference slot and containing the attributes referenced across time slices, and make classes $\mathcal{B}_1$ and $\mathcal{B}_\rightarrow$ implement this interface (Figure 3c). Then, creating a reference slot ρ in $\mathcal{B}_\rightarrow$ whose range is $\mathcal{I}$ allows instances of $\mathcal{B}_\rightarrow$ to reference random variables of the instances of $\mathcal{B}_1$ in the first time slice and those of $\mathcal{B}_\rightarrow$ in the other slices.

As we shall see in Section 4, interfaces will allow us to prove that determining the optimal set of patterns in a relational skeleton that shall be converted into classes to speed-up inference is a NP-hard problem. But before proving this result, we shall recall how inference is performed in PRMs.



$\mathcal{B}_1 \blacktriangleleft \mathcal{I}$
$\mathcal{I}$
$\mathcal{B}_{\rightarrow} \blacktriangleleft \mathcal{I}$
ρ
X^0
X^1
X^2

Figure 4: Interface $\mathcal{I}$ is the solution for modeling dBN as PRM.

3. Structured Inference

Determining the probabilities of a set of random variables given evidence is the most common query performed in probabilistic graphical models. There exists a wide variety of inference algorithms to compute these distributions [1, 27, 28, 29, 30, 31]. They often rely in some way to a Variable Elimination scheme [30, 31]. The basic idea consists of marginalizing out random variables one by one from the joint distribution (or more generally from a set of CPTs or *potentials*) until there only remains the variables of interest, as shown in Algorithm 1. Algorithm 2 shows another inference method, called a *Junction Tree* inference, based on a similar idea. Many efficient variants like Lazy Propagation [31] actually follow the same scheme. 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 $\mathbf{e}$.

Algorithm 1: Variable Elimination (VE).

```
  Input: a set of potentials $\mathbf{P}$ and a set of target random variables $\mathbf{X}$
  Output: $P(\mathbf{X})$
1 $\mathbf{W} \leftarrow$ all the variables of the potentials of $\mathbf{P}$ except $\mathbf{X}$
2 while $\mathbf{W} \neq \emptyset$ do
3     let $X_j$ be some variable in $\mathbf{W}$;  remove $X_j$ from $\mathbf{W}$
4     let $\mathbf{Q}$ be the set of tables in $\mathbf{P}$ containing variable $X_j$
5     compute potential $q = \sum_{X_j} \prod_{f \in \mathbf{Q}} f$
6     $\mathbf{P} \leftarrow (\mathbf{P} \backslash \mathbf{Q}) \cup \{q\}$
7 return potential $\prod_{f \in \mathbf{P}} f$
```

The above scheme is efficient and can be used in PRMs by applying it on their grounded BN. However, by processing random variables separately,

Algorithm 2: Junction tree inference (JT).

```
Input: a set of potentials $\mathbf{P}$ and a set of target random variables $\mathbf{X}$
Output: $P(\mathbf{X})$
1  create a Markov network $G = (\mathbf{V}, \mathbf{E})$ whose nodes represent the random
   variables of the potentials of $\mathbf{P}$ and such that $(X_i, X_j) \in \mathbf{E}$ if and only if
   there exists a potential in $\mathbf{P}$ containing both variables $X_i$ and $X_j$
2  triangulate $G$ by *eliminating* all the nodes in $\mathbf{V} \backslash \mathbf{X}$ and, then, by creating a
   clique for the nodes in $\mathbf{X}$
3  create a junction tree $T = (\mathbf{C}, \mathbf{F})$ from the triangulated graph $G$
4  put each potential of $\mathbf{P}$ in any clique of $\mathbf{C}$ containing all its variables
5  let $R$ be the clique of $T$ corresponding to the set of nodes $\mathbf{X}$
6  call collect($T$,$R$,$R$)  // see Algorithm 3
7  let $\mathbf{Q}$ be the set of tables in clique $R$ plus those in the messages sent from
   the neighbors of $R$ toward $R$
8  let $\mathbf{W}$ be the set of variables in the potentials in $\mathbf{Q}$
9  compute potential $q = \sum_{X_i \in \mathbf{W} \backslash \mathbf{X}} \prod_{f \in \mathbf{Q}} f$
10 return potential $q$
```

VE and the other aforementioned inference algorithms are unable to exploit the structural repetitions in the graphical model to avoid computation redundancies. The aim of Structured Inference is to fill this gap [24, 5] and Object-Oriented frameworks provide a simple and effective way to achieve this goal [32]. 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 prior to the elimination of any other random variable produces precisely the same computations for all the

Algorithm 3: The `collect` phase of the junction tree algorithm.

```
Input: a junction tree $T$, a current clique $C_i$, a previous clique $C_j$
1  foreach clique $C_k \neq C_j$ such that $C_k$ is a neighbor of $C_i$ do
2      call collect($T, C_k, C_i$)
3  if $C_j \neq C_i$ then
4      let $\mathbf{Q}$ be the set of tables in clique $C_i$ plus those in the messages sent
       from $C_i$'s neighbors $C_k \neq C_j$ toward $C_i$
5      let $\mathbf{W}$ be the set of variables in the potentials in $\mathbf{Q}$
6      let $\mathbf{S}$ be the set of nodes belonging to both $C_i$ and $C_j$
7      compute potential $q = \sum_{X_i \in \mathbf{W} \backslash \mathbf{S}} \prod_{f \in \mathbf{Q}} f$
8      send a message containing potential $q$ from $C_i$ to $C_j$
```

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 (see [25] for further details).

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 $\textsc{Range}(\mathcal{F}.\rho).Y$ is a non-resident attribute of $\mathcal{F}$. Class-level elimination corresponds to the elimination of all the inner attributes (using any inference algorithm) within the class itself. 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. This operation is called *class-level elimination.* 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). Once class-level elimination has been performed, Structured Inference goes on with the attributes elimination process at the instance level, i.e., on the resulting grounded BN with any classical inference engine. For instance,

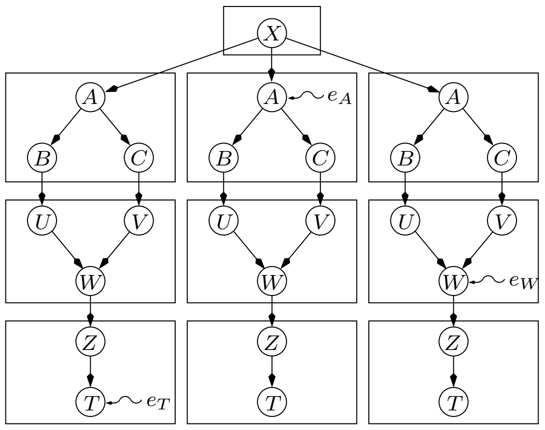


Figure 5: A BN with 4 classes: each set of variables inside a rectangle corresponds to an instance. Some nodes received evidence e_A, e_T, e_W.

consider the PRM/BN of Fig. 5: it is composed of 4 different classes and an overall of 10 instances (represented as rectangles). Nodes A, U, V, Z are the internal nodes that should be removed at class-level. Some nodes received evidence (denoted as e_A, e_T and e_W). At class level, the internal attributes of the instances that received no evidence are removed. This leads to the BN of Fig. 6, on which any inference algorithm (like VE or JT) can be applied. SVE can be formally defined as Algorithm 4. In this algorithm, VE is used only on Line 16. Substituting it by a call to JT (Algorithm 2) will thus produce a junction tree-based structured inference.

As stated in [25], it is worthwhile to emphasize that even if this section essentially focused on the description of a structured VE as a prototype for Bayesian inference in order to ease the presentation, Structured Inference can be used with many other inference algorithms ([27], [28], [30], [31], etc.). Indeed, the ideas for caching intermediate computations between equivalent topological patterns also applies to any inference algorithm that runs on the graphs of the BNs or on their secondary structures (like junction trees).

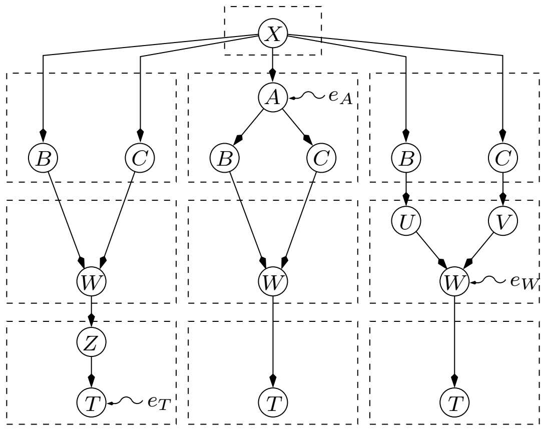


Figure 6: The BN resulting from class-level elimination.

4. PRM's Patterns Discovery: Problem and Complexity

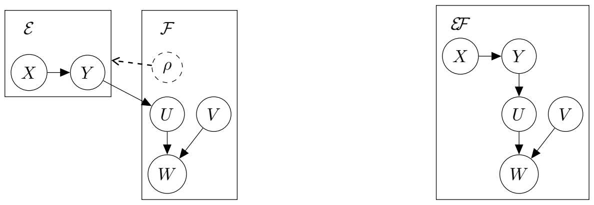
Marginalizing-out internal nodes at *class level* is the key to Structured Inference's efficiency as it reduces significantly redundant computations. However, from a theoretical point of view, not all redundancies can be identified by this scheme: consider the two classes $\mathcal{E}$ and $\mathcal{F}$ of Figure 7.(a) and let $Y \in \mathbf{A}(\mathcal{E})$, $U \in \mathbf{A}(\mathcal{F})$ be two attributes such that the only non-resident child of $\mathcal{E}.Y$ is $\mathcal{F}.U$. Then $\mathcal{E}.Y$ cannot be eliminated at class level because

Algorithm 4: Structured Variable Elimination (SVE).

```
Input: a set of classes $\mathbf{C}$, a system $S$, a set of queried random variables $\mathbf{X}$,
       a set of evidence $\mathbf{e}$
Output: $P(\mathbf{X}|\mathbf{e})$
// compute class-level eliminations
1  foreach $\mathcal{C} \in \mathbf{C}$ do
2      let $\mathbf{I}$ be the set of the inner attributes of $\mathcal{C}$
3      $\mathbf{P}'(\mathcal{C}) \leftarrow \mathbf{P}(\mathcal{C})$
4      foreach attribute $A \in \mathbf{I}$ do
5          let $\mathbf{Q}$ be the set of tables in $\mathbf{P}'(\mathcal{C})$ containing attribute $A$
6          compute potential $q = \sum_A \prod_{f \in \mathbf{Q}} f$
7          $\mathbf{P}'(\mathcal{C}) \leftarrow (\mathbf{P}'(\mathcal{C}) \backslash \mathbf{Q}) \cup \{q\}$
// Compute the instance-level pool of potentials
8  $\mathbf{P} \leftarrow \emptyset$
9  foreach instance $c \in S$ do
10     let $\mathcal{C}$ be the class of instance $c$
11     if there exists a random variable $A \in c$ such that $A \in \mathbf{X}$ or $A$ received
       some evidence $e_A \in \mathbf{e}$ then
12         $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{P}(C)$
13     else
14         $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{P}'(C)$
15 $\mathbf{P} \leftarrow \mathbf{P} \cup_{e_A \in \mathbf{e}} \{P(e_A|A)\}$ // add evidence probabilities to the pool
16 return Variable Elimination($\mathbf{P}, \mathbf{X}, \mathbf{e}$)
```

it is not internal. However, if we consider the "new" class $\mathcal{EF}$ of Figure 7.(b) defined by compound $(\mathcal{E}, \mathcal{F})$, attribute $\mathcal{EF}.Y$ is no longer an outer attribute since $U \in \mathbf{A}(\mathcal{EF})$. Hence, pairs of instances (e, f) of $\mathcal{E}$ and $\mathcal{F}$ that fit the definition of $\mathcal{EF}$ can be considered as instances of $\mathcal{EF}$. In these new instances, Y is internal and, thus, is eligible to class-level elimination.

From a practical perspective, such situations occur frequently in complex systems modeled by PRMs. Indeed, experts usually encode as classes very generic knowledge and, as such, many relations among classes that are specific to a given problem are not taken into account by Structured Inference. For instance, in genetics, a class often reflects the probabilistic genetic relationships between parents and one of their children, and those between grand-parents and children are modeled using one instance of the aforementioned class to represent the relationships between grand-parents and parents, and another one to represent those between parents and child. Therefore, by identifying the situations where such pairs of instances ap-



(a) Original classes $\mathcal{E}$ and $\mathcal{F}$.

(b) Compound $\mathcal{EF}$.

Figure 7: Compound classes.

pear frequently in the relational skeleton, and by creating the corresponding compound classes, Structured Inference can be significantly improved since the opportunities for class-level eliminations are increased. The rest of the section aims at determining such compound classes.

Once compound classes have been identified, transforming the instances of the relational skeleton into instances of these new compound classes must be performed carefully. Indeed, consider again classes $\mathcal{E}, \mathcal{F}, \mathcal{EF}$ of Figure 7, then not all pairs (e, f) of $\mathcal{E}, \mathcal{F}$ are eligible for these transformations: 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 relational skeleton.

In order to find effective compounds/instance-class reassignments, it is most convenient to search them in the following graph:

Definition 6 (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 $(c, d) \in \boldsymbol{\mathcal{E}}$ iff*

$$\mathbf{L}_{cd} = \bigcup_{X \in \mathbf{A}(c)} (\Pi_{c.X} \cap \mathbf{A}(d)) \cup \bigcup_{X \in \mathbf{A}(d)} (\Pi_{d.X} \cap \mathbf{A}(c)) \neq \emptyset. \qquad (1)$$

Edge (c, d) is labeled by $\mathbf{L}_{cd}$.

Fig. 8 illustrates a boundary graph for which different compound classes will be mined. Each circle node corresponds to an instance of the relational skeleton. 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

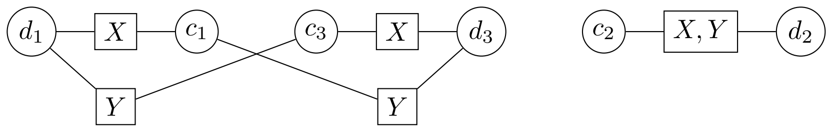


Figure 8: A boundary graph. The different possible connections between instances result in different edge labels (square nodes). We can see that compound $\{c_1, d_1\}$ is different from compound $\{c_2, d_2\}$.

as an instance of a compound. Actually, the first union in Eq. (1) lists the attributes of instance d that are referenced by c and the second union lists those of c referenced by d. Therefore, 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 $\mathbf{L}_{c_1 d_1} \neq \mathbf{L}_{c_2 d_2}$. For instance, in Figure 8, Y is an outer attribute for compound $\{c_1, d_1\}$ whereas it is an inner attribute for compound $\{c_2, d_2\}$. Hence the two pairs of instances shall be treated differently by Structured Inference since their class level eliminations clearly differ. This suggests the following definition to identify which sets of instances should be treated similarly:

Definition 7 (dynamic class). *Let $\mathcal{BG}$ be a boundary graph. A dynamic class $\hat{\mathcal{F}}$ in $\mathcal{BG}$ 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 $\hat{\mathcal{F}}$'s boundary.*

Hence, given a dynamic class $\hat{\mathcal{F}}$, all the nodes in $\mathbf{A}(\hat{\mathcal{F}})\backslash\mathbf{B}$ 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 subgraph by one instance of its dynamic class and, finally, apply a structured 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. As mentioned before, in Fig. 1, the dynamic class resulting from the aggregation of classes $\mathcal{E}$ and $\mathcal{F}$ has three possible instances: (e_1, f_1), (e_1, f_2) and (e_2, f_3). However, (e_1, f_1) and (e_1, f_2) cannot be both considered as instances of this new dynamic class else e_1 would appear twice in the grounded BN, hence resulting in incorrect computations. Consequently, we shall enforce 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:

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 to eliminate all random variables is smaller than K?*

The proof is given in the appendix. 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 [33, 34]. In addition, determining all the occurrences of a given subgraph in a graph is NP-hard as well [35]. 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 [35]. However, the proof of Proposition 1 shows that finding the best dynamic classes/substitutions remains NP-hard even in cases where all the above problems can be solved polynomially, in particular when inference in the grounded BN is polynomial (singly-connected BNs). We shall however present in the next section an efficient approximate algorithm for determining an effective set of dynamic classes.

5. A PRM's Patterns Discovery 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 [36, 37, 38]. 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 [38]. We will present the generic derived algorithm in the next subsection and, then, we will describe an efficient pruning rule that can speed-up significantly this algorithm.

5.1. A gSpan-Based Approximate Algorithm

The basic idea of the algorithm consists of creating a search tree $\mathbb{T}$ that, when fully constructed, contains all the dynamic classes that can be found in the boundary graph as well as their substitutions. Then there just remains to parse efficiently this tree to determine the best set of dynamic classes and boundary subgraph substitutions that satisfy Rule 1. More formally, we define the possible substitutions as:

Definition 8 (Matches of a dynamic class). *Let $\boldsymbol{\mathcal{BG}} = (\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{E}})$ be a boundary graph and let $\widehat{\mathcal{D}}$ be a dynamic class. The matches of $\widehat{\mathcal{D}}$, denoted as $\mathbf{M}(\widehat{\mathcal{D}})$, is the set of sets of the instances of $\boldsymbol{\mathcal{I}}$ such that each $s \in \mathbf{M}(\widehat{\mathcal{D}})$ is a set of instances matching $\widehat{\mathcal{D}}$, i.e., the set of instances in s can be substituted in $\boldsymbol{\mathcal{I}}$ by one single instance d of $\widehat{\mathcal{D}}$. In such a case, d is called the* instance substitution *of s.*

For instance, in Fig. 8, the dynamic class, say $\widehat{\mathcal{F}}$, defined as compound $(\mathcal{C}, \mathcal{D})$ with a boundary equal to $\{\mathcal{C}.Y, \mathcal{D}.Y\}$, has a set of matches equal to $\{\{c_1, d_1\}, \{c_3, d_3\}\}$. Thus, pairs of instances (c_1, d_1) and (c_3, d_3) can both be substituted by one instance of $\widehat{\mathcal{F}}$ without altering the grounded BN.

Let $\boldsymbol{\mathcal{BG}} = (\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{E}})$ be a boundary graph and let $\{\widehat{\mathcal{D}}_1, \ldots, \widehat{\mathcal{D}}_n\}$ be a set of dynamic classes. Denote as $\{m^1_{\widehat{\mathcal{D}}_i}, \ldots, m^{k_{\widehat{\mathcal{D}}_i}}_{\widehat{\mathcal{D}}_i}\}$ the elements of the set of matches $\mathbf{M}(\widehat{\mathcal{D}}_i)$ of dynamic class $\widehat{\mathcal{D}}_i$. Define a graph $G = (V, E)$ where, to each set $m^h_{\widehat{\mathcal{D}}_i}$ is associated a node in V (which we shall call $m^h_{\widehat{\mathcal{D}}_i}$ as well since it is unambiguous), and $E = \{(m^h_{\widehat{\mathcal{D}}_i}, m^l_{\widehat{\mathcal{D}}_j})$: there exists an instance in $\boldsymbol{\mathcal{I}}$ belonging to both $m^h_{\widehat{\mathcal{D}}_i}$ and $m^l_{\widehat{\mathcal{D}}_j}\}$. Then:

Proposition 2. *Any independent set*[2] *of G defines a set of instance substitutions satisfying Rule 1.*

[2]Recall that an independent set of a graph $G = (V, E)$ is a subset $W \subseteq V$ such that no pair of nodes of W are adjacent in G.

Of course, some substitutions are better than others because they induce higher speed-ups in Structured Inference. So the nodes of V should be weighted according to the speed-up improvements they induce. We shall see below how this can be estimated. The "*best*" substitutions we look for then correspond to solutions of a *Max Weighted Independent Set* problem. Unfortunately, solving exactly this problem is NP-hard [35]. Nevertheless, in practice, there exist some algorithms that approximate it quite efficiently [39, 40]. In our experiments, we used the approach advocated in [40].

There now remains to determine the set of dynamic classes $\{\widehat{\mathcal{D}}_1, \dots, \widehat{\mathcal{D}}_n\}$ and their matches that are to be given as input to the max independent set problem. For this purpose, we use a variant of the gSpan pattern mining algorithm [38]. As our goal is to create compound classes, we shall only mine frequent connected boundary subgraphs. This is precisely what gSpan is designed for. To describe the mining of these subgraphs, we define the subgraph of the boundary graph corresponding to a dynamic class:

Definition 9 (Boundary subgraph of a dynamic class). *Let $\boldsymbol{\mathcal{BG}} = (\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{E}})$ be a boundary graph and let $\widehat{\mathcal{D}}$ be a dynamic class compounding some classes $\mathcal{C}_1, \dots, \mathcal{C}_k$ with a boundary equal to $\{B_1, \dots, B_r\}$. The boundary subgraph of $\widehat{\mathcal{D}}$, denoted by $BG(\widehat{\mathcal{D}})$, is the generic subgraph $G = (V, E)$ of $\boldsymbol{\mathcal{BG}}$ such that there exists one node in V per class $\mathcal{C}_i$. The edges of E link all the nodes of V as they are in the boundary graph and, for each B_i, $i = 1, \dots, r$, there exists an edge (X, Y) with X the node of V corresponding to the class containing B_i and Y a dummy node. Edges whose both extremal nodes belong to V are called internal, else they are called external.*

For instance, in Fig. 8, the boundary subgraph of dynamic class $\widehat{\mathcal{F}}$ defined as compound $(\mathcal{C}, \mathcal{D})$ with a boundary equal to $\{\mathcal{C}.Y, \mathcal{D}.Y\}$ is illustrated on Fig. 9. Now, gSpan creates a tree $\mathbb{T}$ whose nodes correspond to dynamic classes. A node $N(\widehat{\mathcal{D}}_i)$ of this tree actually represents a pair $(BG(\widehat{\mathcal{D}}_i), \mathbf{M}(\widehat{\mathcal{D}}_i))$ where $\widehat{\mathcal{D}}_i$ is a dynamic class and $\mathbf{M}(\widehat{\mathcal{D}}_i)$ is the set of matches of $\widehat{\mathcal{D}}_i$. Tree $\mathbb{T}$ is initialized with all the dynamic classes whose boundary subgraphs have 1 internal edge. In $\mathbb{T}$, nodes at level $k+1$ are derived from those at level k by extending their boundary subgraph with one of their adjacent node in $\boldsymbol{\mathcal{BG}}$, i.e., at least one of their external edge becomes internal. This guarantees that the nodes of $\mathbb{T}$ represent dynamic classes with nonempty matches. For

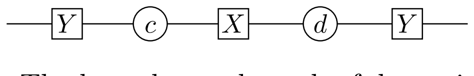
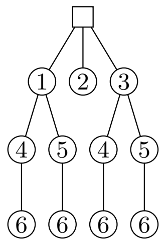


Figure 9: The boundary subgraph of dynamic class $\widehat{\mathcal{F}}$.

instance, the search tree $\mathbb{T}$ for the boundary graph of Fig. 8 is displayed in Fig. 10: each node represents a dynamic class or, equivalently, a specific subgraph of $\boldsymbol{\mathcal{BG}}$. The whole tree thus reveals precisely all the possible connected dynamic classes and instance substitutions that can be applied to the PRM. Fig. 11 illustrates a general tree. Once $\mathbb{T}$ is constructed, its nodes can be exploited to feed the aforementioned max weighted independent set problem in order to get, if not an optimal, at least a good set of instance substitutions.



①	$\widehat{\mathcal{E}} = \langle(\mathcal{C},\mathcal{D}),\{\mathcal{C}.Y,\mathcal{D}.Y\}\rangle$	$\mathbf{M}(\widehat{\mathcal{E}}) = \{\{c_1,d_1\},\{c_3,d_3\}\}$
②	$\widehat{\mathcal{F}} = \langle(\mathcal{C},\mathcal{D}),\emptyset\rangle$	$\mathbf{M}(\widehat{\mathcal{F}}) = \{\{c_2,d_2\}\}$
③	$\widehat{\mathcal{G}} = \langle(\mathcal{C},\mathcal{D}),\{\mathcal{C}.X,\mathcal{D}.X\}\rangle$	$\mathbf{M}(\widehat{\mathcal{G}}) = \{\{c_1,d_3\},\{c_3,d_1\}\}$
④	$\widehat{\mathcal{H}} = \langle(\mathcal{D}_1,\mathcal{C},\mathcal{D}_3),\{\mathcal{D}_1.X,\mathcal{D}_3.Y\}\rangle$	$\mathbf{M}(\widehat{\mathcal{H}}) = \{\{d_3,c_1,d_1\},\{d_3,c_3,d_1\}\}$
⑤	$\widehat{\mathcal{J}} = \langle(\mathcal{C}_1,\mathcal{D},\mathcal{C}_3),\{\mathcal{C}_1.X,\mathcal{C}_3.Y\}\rangle$	$\mathbf{M}(\widehat{\mathcal{J}}) = \{\{c_3,d_1,c_1\},\{c_3,d_3,c_1\}\}$
⑥	$\widehat{\mathcal{K}} = \langle(\mathcal{C}_1,\mathcal{D}_1,\mathcal{C}_3,\mathcal{D}_3),\emptyset\rangle$	$\mathbf{M}(\widehat{\mathcal{K}}) = \{\{c_1,d_1,c_3,d_3\}\}$

Figure 10: Search tree $\mathbb{T}$ for the boundary graph of Fig. 8.

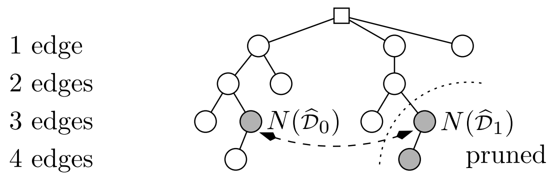


Figure 11: Dynamic classes search tree $\mathbb{T}$.

Of course, the size of $\mathbb{T}$ is exponential and, thus, some pruning is necessary to get a fast approximate algorithm. The most obvious pruning rule consists of removing duplicate nodes in the search tree. Indeed, the children 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 dynamic class $\widehat{\mathcal{D}}_1$ is the same 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 tree (see Fig. 11). Such a case arises for instance when $\widehat{\mathcal{D}}_0$ is constructed by first compounding some classes $\mathcal{C}$ and $\mathcal{D}$, and then compounding the result with some class $\mathcal{E}$, while $\widehat{\mathcal{D}}_1$ is constructed by first compounding $\mathcal{D}$ with $\mathcal{E}$, and then compounding the result with $\mathcal{C}$ (see Fig. 10 for an example).

Determining whether two subgraphs are identical is known as the *subgraph isomorphism* problem and is NP-complete in the general case [41].

However, in practice, there exist efficient algorithms for solving this task [42, 43]. In gSpan, this is achieved using a DFS canonical labeling of subgraphs defined as follows: given a boundary subgraph $BG(\widehat{c}) = (V, E)$, a depth first search (DFS) tree T is created to parse all the nodes in V. The depth-first discovery of the nodes induces a linear order on V and the latter induces a linear order $\succ_{E,T}$ on E. The label or DFS code of $BG(\widehat{c})$ w.r.t. T is the set of edges E sorted in increasing order w.r.t. $\succ_{E,T}$. In gSpan, the nodes of V are supposed to be labeled and it is assumed that there exists a linear order $\succ_L$ on those labels. In our case, the classes of the PRM have a unique identifier which is assigned as label to each instance of the class. Using $\succ_{E,T}$ and $\succ_L$, gSpan defines a lexicographic order on the codes of $BG(\widehat{c})$. The minimal one w.r.t. to this order is the canonical label of the $BG(\widehat{c})$. It is shown in [38] that two graphs are isomorphic if and only if they have the same canonical label.

At first sight, this mechanism seems a heavy machinery since, in theory, we shall need to compute all the DFS trees of $BG(\widehat{c})$ in order to get all the codes of $BG(\widehat{c})$ and select the minimal one. Fortunately, it is shown in [38] how the DFS code assigned to a node $N(\widehat{c})$ of search tree $\mathbb{T}$ can be extended to form the prefix of the code of any of its children, say $N(\widehat{\mathcal{D}})$. This imposes a restriction on $N(\widehat{\mathcal{D}})$: let T be the DFS tree generating the canonical label of $\widehat{c}$ and assume that $BG(\widehat{c}) = (V, E)$. As mentioned previously, T generates a linear order on the nodes of V. The path between the first and the last node of V w.r.t. this order is called the *right path* of $BG(\widehat{c})$ and it is shown in [38] that the children of $BG(\widehat{c})$ in $\mathbb{T}$ can only be formed by adding to $BG(\widehat{c})$ an edge whose extremal nodes belonging to V also belong to the right path. By adding only those edges, we get a fast mechanism for determining DFS codes but this also speeds-up the mining algorithm by reducing the number of duplicate dynamic classes in $\mathbb{T}$.

The above paragraphs describe how the whole search tree $\mathbb{T}$ can be generated. However, as mentioned before, to get a fast algorithm, some pruning rules are necessary. To guaranty their efficiency, we shall construct $\mathbb{T}$ in such a way that the "best" dynamic classes are constructed first. For this purpose, linear order $\succ_L$ is defined so that the more promising the class for compounds the smaller its index in $\succ_L$. Estimating whether a node is promising is simply done by estimating the number of operations saved if its class is part of a dynamic class. Then nodes of each level of $\mathbb{T}$ are sorted according to this order [38]. Thus, parsing $\mathbb{T}$ in a depth-first search (DFS) manner guarantees that the "most promising" dynamic classes are constructed first. The whole process leads to Algorithm 5.

Algorithm 5: Computing dynamic classes/substitutions.

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

We will now describe the pruning rule that, besides the one removing the duplicates, allowed us to prune significantly the search tree.

5.2. Pruning Rules

The rule we used is related to the gain achievable in Structured Inference using dynamic classes: nodes $N(\widehat{\mathcal{D}})$ that define classes whose instance substitutions do not speed-up Structured Inference can be pruned. To estimate the gain in speed, recall that, by Rule 1, only a subset $S_{\widehat{\mathcal{D}}}$ of $\mathbf{M}(\widehat{\mathcal{D}})$ can be substituted in $\mathcal{BG}$ by instances of $\widehat{\mathcal{D}}$. The number of operations (multiplications, additions) performed by Structured Inference on these instance substitutions is equal to:

$$|\text{Comp}(S_{\widehat{\mathcal{D}}})| = w_{\widehat{\mathcal{D}}} + |S_{\widehat{\mathcal{D}}}| \times \overline{w}_{\widehat{\mathcal{D}}},$$

where $|S_{\widehat{\mathcal{D}}}|$ denotes the cardinal of set $S_{\widehat{\mathcal{D}}}$ and 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 $BG(\widehat{\mathcal{D}})$ corresponds to a 1-edge extension of the boundary subgraph of its parent $\pi(\widehat{\mathcal{D}})$ in tree $\mathbb{T}$. So, the matches of $\mathbf{M}(\widehat{\mathcal{D}})$ that were not substituted, i.e., those of $R_{\widehat{\mathcal{D}}} = \mathbf{M}(\widehat{\mathcal{D}}) \backslash S_{\widehat{\mathcal{D}}}$, have a boundary subgraph which is an edge extension of the boundary subgraph $BG(\pi(\widehat{\mathcal{D}}))$. Assuming that all the elements of $R_{\widehat{\mathcal{D}}}$ were substituted as instances of $\pi(\widehat{\mathcal{D}})$, their eliminations by Structured Inference would have the following cost:

$$|\text{Comp}(R_{\widehat{\mathcal{D}}})| = w_{\pi(\widehat{\mathcal{D}})} + (|\mathbf{M}(\widehat{\mathcal{D}})| - |S_{\widehat{\mathcal{D}}}|) \times \overline{\overline{w}}_{\widehat{\mathcal{D}}}$$

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:

$$|\mathrm{Comp}(\mathbf{M}(\widehat{\mathcal{D}}))| = w_{\widehat{\mathcal{D}}} + w_{\pi(\widehat{\mathcal{D}})} + |S_{\widehat{\mathcal{D}}}| \times \overline{w}_{\widehat{\mathcal{D}}} + (|\mathbf{M}(\widehat{\mathcal{D}})| - |S_{\widehat{\mathcal{D}}}|) \times \overline{\overline{w}}_{\widehat{\mathcal{D}}}$$

whereas, by just exploiting $\pi(\widehat{\mathcal{D}})$, it would have been

$$|\mathrm{Comp}'(\mathbf{M}(\widehat{\mathcal{D}}))| = w_{\pi(\widehat{\mathcal{D}})} + |\mathbf{M}(\widehat{\mathcal{D}})| \times \overline{\overline{w}}_{\widehat{\mathcal{D}}}.$$

So, class $\widehat{\mathcal{D}}$ is unattractive for inference and $N(\widehat{\mathcal{D}})$ may be pruned whenever:

$$\alpha_{\widehat{\mathcal{D}}} = |\mathrm{Comp}(\mathbf{M}(\widehat{\mathcal{D}}))| - |\mathrm{Comp}'(\mathbf{M}(\widehat{\mathcal{D}}))| = w_{\widehat{\mathcal{D}}} + |S_{\widehat{\mathcal{D}}}| \times (\overline{w}_{\widehat{\mathcal{D}}} - \overline{\overline{w}}_{\widehat{\mathcal{D}}}) > 0.$$

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/approximating a *Max Independent Set* problem induced by $\mathbf{M}(\widehat{\mathcal{D}})$. To estimate $w_{\widehat{\mathcal{D}}}$, it is sufficient to compute a junction tree of $\widehat{\mathcal{D}}$'s DAG [44, 45], eliminating only inner nodes, and to sum-up the sizes of its cliques. Incremental triangulations are particularly fast to perform this task [46]. 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.

6. 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 [32], and also with Variable Elimination (VE), a classic and standard probabilistic inference algorithm for Bayesian Networks [30].

As explained in section 3, VE is used here as a prototypical inference algorithm and we could have used other inference algorithms for the experimentations. VE is peculiar among the probabilistic inference algorithms in that it mainly focuses on the computation of the posterior marginal distribution of a single variable. One can roughly say that inference algorithms often require two phases: i) collect all observations and ii) distribute them. In this general scheme, VE would only correspond to the first phase. As the second phase would also benefit from structured inference, using another algorithm would just increase the gains for structured inference. In that sense, using VE is a worst case for our algorithm.

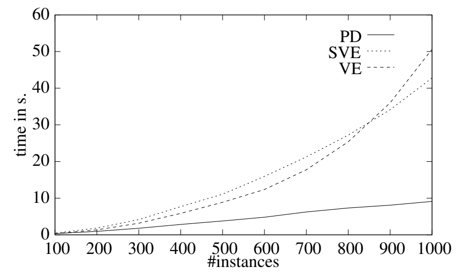
It is important to note that we did not need to include evidence in our experiments. This choice was motivated by the fact that the structure of the network may vary drastically in the presence of evidence while our goal here was to show how pattern mining can improve inference when there exist repetitions in the network. Indeed, evidence are not a good indicator of repetitions as they can either be entered at the same location in every pattern, thus preserving repetitions, or be entered randomly, thus breaking the structure. This is why experiments 1 and 2 show the results of our new approach on networks with and without repetitions rather than with or without evidence, hence providing a much better insight of PD's performance in all possible situations.

Response times reported for PD take into account both pattern mining and inference. For experiments using VE, results include both grounding and inference time. All our experiments were performed on a 2.7 Ghz Intel Xeon. The source code of our PRMs implementation, the inference algorithm and the generation algorithms can be found in the aGrUM project[3].

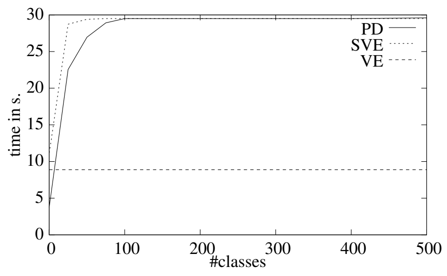
6.1. PRM random generation

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 repetitions 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.

[3]`http://agrum.lip6.fr`



(a) First set of experiments.



(b) Second set of experiments.

Figure 12: Structural repetition is an important factor for PD's performance. Unsurprisingly, performance decreases dramatically for systems with no structural repetition.

The PRM's generation process is performed as follows: first, we generate an interface $\mathcal{I}$[4] with min_{attr} attributes which will be implemented by all classes 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 in $[0, max_{ref}]$. Finally, we generate a DAG S representing the relational skeleton of our 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 $\mathcal{C}$ is generated as follows: we first create a DAG $G_{\mathcal{C}}$ with max_{attr} nodes, we then add to $\mathcal{C}$ k reference slots and max_{attr} attributes. Dependencies between attributes are defined using $G_{\mathcal{C}}$. 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 $\mathcal{C}$ chosen randomly. DAGs are generated using the algorithm provided in [47].

6.2. Experimental results

In our first set of experiments, we generated systems with an increasing number of instances. Each class contains 15 attributes ($max_{attr} = 15$) and has at most 4 incoming arcs ($max_{ref} = 4$). Each attribute's domain size

[4]Remember that if a class implements a given interface, then this guarantees the existence of the attributes and reference slots defined in that interface [23].

is equal to 2 (*domain* $= 2$). This choice was made to penalize as much as possible our algorithm against the other methods. As a matter of fact, PD's mining time is unaffected by the domain size of the random variables and the inference improvement due to dynamic classes increases with it. 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. 12a 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 repetitions 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 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. As stated below, these experiments are also a fair presentation of the behavior of our algorithms in presence of randomly chosen evidence. Actually, entering evidence into a BN amounts to make the instances that received them belong to "new classes" that correspond to classes whose CPTs include precisely the probabilities of these evidence. As such, experiments in which we increase the number of classes are thus a good representative of inference times in BNs in which the number of random variables that received evidence increase.

Fig. 12b 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 VE's response times, we see that PD and SVE are considerably counter-performing. To anyone familiar with structural inference, this is an unsurprising result which can be explained by the fact that the elimination order used by PD and SVE (inner attributes are eliminated before outer ones) is in most cases suboptimal. If PD and SVE show better results than VE in Fig. 12a it is only because the gain resulting from the reduction of redundant computations more than 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 can also be an 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 implicitly take into account frequencies and cut the mining process when such value is too low.

In our third experiment, we analyze the amount of patterns found by PD with the parameters from experiment 1 ($max_{attr} = 15, domain = 2, max_{ref} = 4, c = 5, max_{ref} + 1 = 5$). The results of this experiment are summarized in Table 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

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. repetition	max pat. repetition	#inst. per pat.	max inst. per pat.	% of attr. 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%

the main patterns were applied. If we consider the last column of Table 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[5]. 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.

7. 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 PRMs. 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.

[5]see `http://www.cs.huji.ac.il/~galel/Repository/`

8. Appendix: proofs

Proof of Proposition 1: We provide a reduction from Vertex Cover for cubic graphs, which is known to be NP-complete [48]:
Instance: A graph $G = (V, E)$, whose vertices $V = \{v_1, ..., v_n\}$ and edges $E = \{e_1, ..., e_m\}$ are such that each v_i has 3 neighbors. An integer $K \geq 0$.
Question: Is there a set of vertices $C \subseteq V$ of size at most K such that each e_i is incident to at least one vertex in C?

Definition of a PRM related to the Vertex Cover problem:

Define a PRM as shown in Fig.13: let $H = 8$ and let X and Y be the prototypes of two octary random variables (i.e., the domain sizes of X and Y are equal to H). Let $\mathcal{I}_x = \langle \mathbf{A}(\mathcal{I}_x) = \{X\}, \mathbf{R}(\mathcal{I}_x) = \emptyset \rangle$ and $\mathcal{I}_y = \langle \mathbf{A}(\mathcal{I}_y) = \{Y\}, \mathbf{R}(\mathcal{I}_y) = \emptyset \rangle$ be two interfaces. Create the following classes $\mathcal{S}, \mathcal{T}, \mathcal{Q}, \mathcal{U}_i, \mathcal{R}_{ij}, \mathcal{Z}_{ij}$:

- class $\mathcal{S}$ implements interface $\mathcal{I}_y$. It contains only one attribute $\mathbf{A}(\mathcal{S}) = \{Y\}$, no reference slot $\mathbf{R}(\mathcal{S}) = \emptyset$, a DAG containing only node Y and a set of probability distributions $\mathbf{P}(\mathcal{S}) = \{P_S(Y)\}$;

- class $\mathcal{T}$ implements $\mathcal{I}_x$. It has one attribute $\mathbf{A}(\mathcal{T}) = \{X\}$, one reference slot $\mathbf{R}(\mathcal{T}) = \{\rho\}$ such that $\text{Range}(\mathcal{T}.\rho) = \mathcal{I}_y$, a DAG $Y \to X$ and a set of probability distributions $\mathbf{P}(\mathcal{T}) = \{P_T(X|Y)\}$;

- class $\mathcal{Q}$ implements $\mathcal{I}_y$. It contains one attribute $\mathbf{A}(\mathcal{Q}) = \{Y\}$, one reference slot $\mathbf{R}(\mathcal{Q}) = \{\rho\}$ such that $\text{Range}(\mathcal{Q}.\rho) = \mathcal{I}_x$, a DAG $X \to Y$ and a set of probability distributions $\mathbf{P}(\mathcal{Q}) = \{P_Q(Y|X)\}$;

- for each node $v_i \in V$, class $\mathcal{U}_i$ implements $\mathcal{I}_y$. It contains one attribute $\mathbf{A}(\mathcal{U}_i) = \{Y\}$, one reference slot $\mathbf{R}(\mathcal{U}_i) = \{\rho\}$ such that $\text{Range}(\mathcal{U}_i.\rho) = \mathcal{T}$. Class $\mathcal{U}_i$'s DAG is $X \to Y$ and $\mathbf{P}(\mathcal{U}_i) = \{P_{U_i}(Y|X)\}$;

- for any i, j, class $\mathcal{R}_{ij}$ implements $\mathcal{I}_y$. It contains one attribute $\mathbf{A}(\mathcal{R}_{ij}) = \{Y\}$ and one reference slot $\mathbf{R}(\mathcal{R}_{ij}) = \{\rho\}$ such that $\text{Range}(\mathcal{R}_{ij}.\rho) = \mathcal{T}$. Class $\mathcal{R}_{ij}$'s DAG is $X \to Y$ and $\mathbf{P}(\mathcal{R}_{ij}) = \{P_{R_{ij}}(Y|X)\}$;

- for any i, j, class $\mathcal{Z}_{ij}$ implements $\mathcal{I}_x$. It contains one attribute $\mathbf{A}(\mathcal{Z}_{ij}) = \{X\}$ and one reference slot $\mathbf{R}(\mathcal{Z}_{ij}) = \{\rho\}$ such that $\text{Range}(\mathcal{Z}_{ij}.\rho) = \mathcal{I}_y$. Class $\mathcal{Z}_{ij}$'s DAG is $Y \to X$ and $\mathbf{P}(\mathcal{Z}_{ij}) = \{P_{Z_{ij}}(X|Y)\}$.

All the distributions $P_S, P_T, P_Q, P_{U_i}, P_{R_{ij}}, P_{Z_{ij}}$, for all i and j, are distinct. Figure 13 illustrates all these classes.

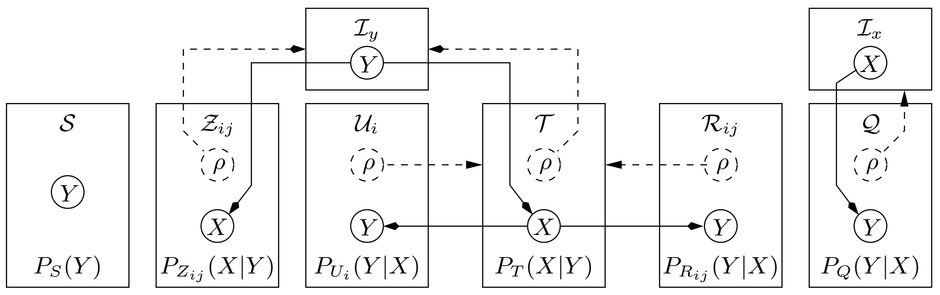


Figure 13: The classes used in the proof's PRM.

Hereafter, for any instances, say a and b, of these classes, ab and $\prod_{i=1}^{n} a$ are shortcuts for subgraphs (or patterns) $a \to b$ and $a \to a \to \cdots \to a$ respectively. For example, if t, q, z are instances of the above classes $\mathcal{T}$, $\mathcal{Q}$ and $\mathcal{Z}_{ij}$ respectively, then tqz is a shortcut for the BN depicted in Fig. 14.a.

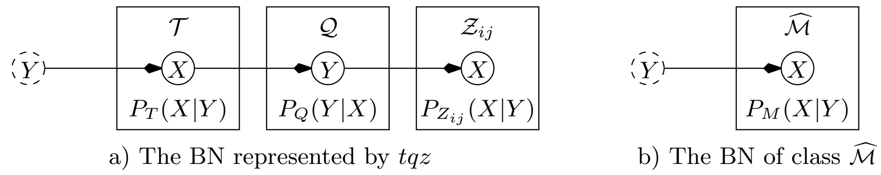


Figure 14: The BN represented by pattern tgz and its dynamic class.

In addition, for any pattern $M = a_1 a_2 \ldots a_r$ of instances, $\widehat{\mathcal{M}}$ denotes a dynamic class implementing a_r's class interface and such that $\mathbf{A}(\widehat{\mathcal{M}})$ contains only the attribute $\in \{X, Y\}$ in $\mathbf{A}(a_r)$, call it b, $\mathbf{R}(\widehat{\mathcal{M}})$ contains only the interface of the reference in a_1's class, call it I_c, $c \in \{X, Y\}$, the DAG of $\widehat{\mathcal{M}}$ is $c \to b$ and $\mathbf{P}(\widehat{\mathcal{M}}) = \{P_M(b|c)\}$, where P_M is the distribution resulting from the elimination of instances $a_1, \ldots, a_{r-1}$ from M. For example, if t, q, z are instances of $\mathcal{T}$, $\mathcal{Q}$ and $\mathcal{Z}_{ij}$ respectively, and if $M = tqz$, then $\widehat{\mathcal{M}}$ is the dynamic class depicted in Fig. 14.b, i.e., the dynamic class such that $\mathbf{A}(\widehat{\mathcal{M}}) = \{X\}$, $\mathbf{R}(\widehat{\mathcal{M}}) = \{\mathcal{I}_y\}$, its DAG is $Y \to X$ and its probability distribution is $\mathbf{P}(\widehat{\mathcal{M}}) = \{P_M(X|Y)\}$, where $P_M(X|Y) = \sum_{X'} \sum_{Y'} P_T(X'|Y) P_Q(Y'|X') P_{Z_{ij}}(X|Y')$. Pattern M can thus be considered as a dynamic class in which we have eliminated all its internal nodes $a_1, \ldots, a_{r-1}$ and in which we have inserted back its interface into the boundary graph. Note that, as M is a chain, computing its probability distribution P_M requires $(r-1)H^3$ operations since each a_i's elimination is of the form $\sum_b P_{a_i}(b|c) P_{a_{i+1}}(c|b)$.

Now, let us construct the PRM's grounded network related to the Vertex

Cover problem: for any $v_i \in V$, let A_i and B_i represent patterns tu_it and qtu_itq respectively, where t, q, u_i are instance prototypes of the classes with the same uppercase names. For any edge $e_i = (v_j, v_k) \in E$, let C_i represent pattern qtu_jtqtu_ktq. The PRM network we will consider in relation to the Vertex Cover problem is defined as follows:

$$BN = s \left(\prod_{j=1}^{40} \prod_{i=1}^{n} (A_i r_{ji}) \right) z_{00} \left(\prod_{j=1}^{15} \prod_{i=1}^{n} (B_i z_{ji}) \right) \prod_{i=1}^{m} (C_i z_{0i}). \qquad (2)$$

As an example, for the instance of Vertex Cover corresponding to the graph of Fig. 15, where the set of nodes is $V = \{v_1, \ldots, v_4\}$ and the set of edges is $E = \{(v_i, v_j) : j > i\}$, Eq. (2) corresponds to:

$$BN = s \prod_{j=1}^{40} (A_1 r_{j1} A_2 r_{j2} A_3 r_{j3} A_4 r_{j4}) z_{00} \prod_{j=1}^{15} (B_1 z_{j1} B_2 z_{j2} B_3 z_{j3} B_4 z_{j4}) \prod_{i=1}^{6} (C_i z_{0i}),$$

where the first two products are related to the nodes of V and the last product to the 6 edges of E. Substituting patterns A_i, B_i, C_i by their instances, we get the following BN:

$$\begin{aligned} BN = s & \textstyle\prod_{j=1}^{40} (tu_1tr_{j1}tu_2tr_{j2}tu_3tr_{j3}tu_4tr_{j4}) z_{00} \\ & \textstyle\prod_{j=1}^{15} (qtu_1tqz_{j1}qtu_2tqz_{j2}qtu_3tqz_{j3}qtu_4tqz_{j4}) \\ & \textstyle\prod_{i=1}^{3} \prod_{j=i+1}^{4} (qtu_itqtu_jtqz_{0i}). \end{aligned}$$

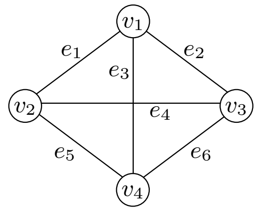


Figure 15: A simple cubic graph.

The proof consists of showing that G has a vertex cover of size at most K if and only if $|\text{Comp}(BN)| \leq \Delta(K)$, where $|\text{Comp}(BN)|$ is the number of operations performed by Structured Inference (SVE) to remove all the nodes in the BN and:

$$\Delta(K) = H + H^2 + \frac{H^3}{4}[78n + K + 2m].$$

1. Proof that G has a vertex cover of size $\leq K \Rightarrow |\mathbf{Comp}(BN)| \leq \Delta(K)$:

First, note that BN is a chain. Hence, using SVE, removing an attribute at the end of the chain induces H^2 operations. Similarly, when all the attributes except s have been removed, removing s requires H operations. Assume that G has a vertex cover C of size $k \leq K$. For each node $v_i \in V$, apply substitution $\widehat{\mathcal{A}}_i$, i.e., substitute all the patterns $A_i = tu_it$ in the first product of Eq. (2) by an instance a_i of $\widehat{\mathcal{A}}_i$ which, by definition (see above), has only one attribute t. Then, the first product of Eq. (2), which originally corresponded to: $\prod_{j=1}^{40} \prod_{i=1}^{n} (tu_itr_{ji})$ is now substituted by $\prod_{j=1}^{40} \prod_{i=1}^{n} (tr_{ji})$, which contains only $80n$ attributes. The cost of these substitutions, i.e., that of class-level elimination, is equal to $n \times 2H^3$ since, in each dynamic class $\widehat{\mathcal{A}}_i$, 2 attributes are eliminated at this level.

For each node $v_i \in C$ (resp. $v_i \in V \backslash C$), apply substitution $\widehat{\mathcal{B}}_i$ (resp. $\widehat{\mathcal{A}}_i$) in the second product of Eq. (2). Thus, the latter is replaced (up to some permutation) by $\prod_{j=1}^{15} \prod_{i:v_i \in C} (b_iz_{ji}) \prod_{i:v_i \in V \backslash C} (qa_iqz_{ji})$. Overall, this product contains $15|C| \times 2 + 15|V \backslash C| \times 4 = 30k + 60(n-k)$ attributes. The class-level elimination cost of dynamic classes $\widehat{\mathcal{B}}_i$ is equal to $k \times 4H^3$ since 4 attributes (qtu_it) are eliminated at this level. Classes $\widehat{\mathcal{A}}_i$ do not incur additional class-level eliminations since those were already performed in the preceding paragraph.

Finally, as C is a vertex cover, each edge of E is incident to a node in C, hence each pattern C_i corresponding to edge (v_j, v_h) can be substituted by either b_ja_hq or qa_jb_h where a_j, a_h and b_j, b_h are instances of dynamic classes $\widehat{\mathcal{A}}_j, \widehat{\mathcal{A}}_h, \widehat{\mathcal{B}}_j, \widehat{\mathcal{B}}_h$ respectively. These substitutions do clearly not incur any additional class-level cost, and they imply that each term of the last product in Eq. (2) contains only 4 attributes (including z_{0i}).

After all these substitutions, BN is a chain with s plus $80n$ attributes in the first product, 1 attribute for z_{00}, $30k + 60(n-k)$ attributes in the second product and $4m$ attributes in the last product. Overall, BN is a chain with s plus $R = 1 + 140n + 4m - 30k$ attributes. As shown at the beginning of the proof, once probabilities P_{A_i} and P_{B_i} of dynamic classes $\widehat{\mathcal{A}}_i$ and $\widehat{\mathcal{B}}_i$ have been computed, the remaining number of computations to eliminate all the attributes is $H + RH^2$. Hence, the overall number of computations, including class-level eliminations, is $2nH^3 + 4kH^3 + H + RH^2 = \Delta(k) \leq \Delta(K)$ operations. Therefore, if G has a vertex cover of size $k \leq K$, then SVE can eliminate all the nodes in BN in $|\text{Comp}(BN)| \leq \Delta(K)$ operations.

2. Proof that $|\mathbf{Comp}(BN)| \leq \Delta(K) \Rightarrow G$ has a vertex cover of size $\leq K$:

Assume now that there exists a set $\widehat{\boldsymbol{\mathcal{M}}}$ of dynamic classes enabling to substitute BN by another instance graph BN' such that the overall number of computations (including class-level eliminations) is less than or equal to $\Delta(K)$. Without loss of generality, we may assume that no substituted pattern contains a node z_{ij} or r_{ij}. Actually, no two z_{ij} or r_{ij} in BN contain the same probability distribution. Therefore, only one substitution of these dynamic classes would be possible in BN. Using these substitutions would thus result in an increase of computations compared to not using them since the elimination of each internal node in dynamic classes requires H^3 operations whereas removing them at instance level from the last attribute to the first one as mentioned above requires only H^2 operations. Therefore, removing all the dynamic classes containing z_{ij} or r_{ij} from set $\widehat{\boldsymbol{\mathcal{M}}}$ produces a new set of dynamic classes inducing an overall number of computations less than or equal to $\Delta(K)$. For the same reason, we may safely assume that substituted patterns do not contain $u_j tqtu_k$ since such pattern also appears at most once in BN (in the last product of Eq. (2)). The rest of the proof consists of showing that including the substitutions described in part 1 of the proof above into $\widehat{\boldsymbol{\mathcal{M}}}$ leads to a set of dynamic classes at least as efficient as $\widehat{\boldsymbol{\mathcal{M}}}$. Then the fact that $|\text{Comp}(BN)| \leq \Delta(K) \Rightarrow G$ has a vertex cover of size $\leq K$ follows.

2.a. $\widehat{\boldsymbol{\mathcal{M}}}$ should contain $\widehat{\mathcal{A}}_i$:

Let i be such that dynamic class $\widehat{\mathcal{A}}_i$ does not belong to $\widehat{\boldsymbol{\mathcal{M}}}$. Then, substituting all patterns A_i by instances of $\widehat{\mathcal{A}}_i$ in the first product of Eq. (2) reduces the overall number of operations in SVE. Actually, in this product, pattern A_i appears 40 times in BN. Removing those occurrences at instance level thus requires $40 \times 3H^2$ whereas exploiting $\widehat{\mathcal{A}}_i$ requires $2H^3$ operations at class level and $40H^2$ operations at instance level. Of course, there may already exist dynamic classes in $\widehat{\boldsymbol{\mathcal{M}}}$ that induced substitutions in the first product of Eq. (2) but, as seen above, these classes do not include r_{ij} and, as such, they can only be equal to either $\widehat{tu_i}$ or $\widehat{u_i t}$, i.e., they correspond to subpatterns of A_i. Whenever one of these two patterns occur, pattern A_i also occurs and the latter eliminates more nodes at class level. Therefore, replacing dynamic classes $\widehat{tu_i}$ and $\widehat{u_i t}$ by $\widehat{\mathcal{A}}_i$ reduces the overall number of computations. $\widehat{\mathcal{A}}_i$ can also be used in the second and third products of Eq. (2) where it was not substituted since this further decreases the inference's number of operations. So we shall safely consider that $\widehat{\boldsymbol{\mathcal{M}}}$ contains $\{\widehat{\mathcal{A}}_i, i = 1, \ldots, n\}$.

2.b. $\widehat{\boldsymbol{\mathcal{M}}}$ should only contain some $\widehat{\mathcal{A}}_i$ and $\widehat{\mathcal{B}}_i$:

It is easy to see that we shall never use $\widehat{qtu_i}$ nor $\widehat{u_i tq}$. Actually, such patterns appear only in the second and third products of Eq. (2). In the second product, removing terms $B_i z_{ji}$ exploiting $\widehat{qtu_i}$ (resp. $\widehat{u_itq}$) requires $2H^3$ class-level operations (removing qt, resp. u_it) and $60H^2$ instance-level operations (since there remains four attributes u_itqz_{ji} or $qtqz_{ji}$). As for the last product, there exists one pattern C_j per edge in graph G and, by definition, each node of G has 3 neighbors. Hence, exploiting $\widehat{qtu_i}$ (resp. $\widehat{u_itq}$) in the last product results in eliminating all qtu_i (resp. u_itq) in $3H^2$ instance-level operations (3 times 1 instance-level operation).

Now, whenever pattern qtu_i or u_itq appears, the larger pattern $B_i = qtu_itq$ also appears. By exploiting $\widehat{\mathcal{B}}_i$, eliminating all the instances b_iz_{ji} in the second product only requires $4H^3$ class-level operations and $30H^2$ instance-level operations. If we do not even use $\widehat{\mathcal{B}}_i$ in the third product of Eq. (2) and eliminate patterns qtu_i (resp. u_itq) at instance level, this adds $9H^2$ instance-level operations, hence an overall of $4H^3$ class-level operations and $39H^2$ instance-level operations, which is less than the number of operations using $\widehat{qtu_i}$ or $\widehat{u_itq}$. Of course, we may use $\widehat{\mathcal{B}}_i$ for the second product and $\widehat{qtu_i}$ or $\widehat{u_itq}$ in the last one, but then, the latter would result in $2H^3$ class-level operations and $3H^2$ instance-level operations instead of $9H^2$ instance-level operations if $\widehat{qtu_i}$ or $\widehat{u_itq}$ were not used in the last product.

Similarly, we shall safely assume that neither $\widehat{qtu_it}$ nor $\widehat{tu_itq}$ belong to $\widehat{\boldsymbol{\mathcal{M}}}$ since they induce $3H^3+45H^2$ operations to eliminate the attributes b_iz_{ji} in the second product of Eq. (2) and can save up to $9H^2$ in the $\prod_{i=1}^m C_i$ part, which is never better than not applying the substitution. Finally, no dynamic class should strictly contain pattern B_i because such a pattern would appear only once in BN and, therefore, its substitution would increase the number of elimination operations. Consequently, we may safely assume that set $\widehat{\boldsymbol{\mathcal{M}}}$ only contains some dynamic classes $\widehat{\mathcal{A}}_i$ and $\widehat{\mathcal{B}}_i$.

2.c. G **has a vertex cover of size** $\leq K$:

To summarize: all A_i are substituted by $\widehat{\mathcal{A}}_i$, some B_i are substituted by $\widehat{\mathcal{B}}_i$ and set $\widehat{\boldsymbol{\mathcal{M}}}$ contains only dynamic classes $\widehat{\mathcal{A}}_i$ and $\widehat{\mathcal{B}}_i$. Therefore, patterns B_i that were not substituted by $\widehat{\mathcal{B}}_i$ are either not substituted at all or are substituted by $q\widehat{\mathcal{A}}_iq$. As the latter decreases the number of computations, we shall assume that the latter case obtains. For the same reason, C_i are substituted by $\widehat{\mathcal{B}}_j\widehat{\mathcal{A}}_hq$ or $q\widehat{\mathcal{A}}_j\widehat{\mathcal{B}}_h$ or $q\widehat{\mathcal{A}}_jq\widehat{\mathcal{A}}_hq$.

Let f and g denote the number of B_i substituted by $\widehat{\mathcal{B}}_i$ and the number of C_i substituted by $q\widehat{\mathcal{A}}_jq\widehat{\mathcal{A}}_hq$ respectively. Then the overall number of computations, including class-level eliminations, is equal to $H + H^2 + \frac{H^3}{4}[78n + f + g + 2m]$. By assumption, this quantity is $\leq \Delta(K)$, hence $f + g \leq K$. Now, as all the variables have been eliminated in product $\prod_{i=1}^m (C_iz_{0i})$, the

f substitutions $\widehat{\mathcal{B}}_j$ have been used to eliminate $f = m - g$ attributes $C_i z_{0i}$. Hence, the nodes corresponding to substitutions in $\widehat{\mathcal{B}}_i$ are adjacent to $m - g$ edges or, equivalently, to all the edges except g edges. By adding one node from each of these edges, we construct a vertex cover of size $f + g \leq K$. ■

Proof of Proposition 2: In an independent set W, there exists no pair of adjacent nodes. Therefore, no pair of nodes $m^h_{\widehat{\mathcal{D}}_i}, m^k_{\widehat{\mathcal{D}}_j}$ in W share an instance of $\boldsymbol{\mathcal{I}}$. Thus the elements of W represent instance substitutions satisfying Rule 1. ■

References

[1] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, 1988.

[2] S. Mahoney, K. Laskey, Network engineering for complex belief networks, in: Proc. of UAI'96, 1996, pp. 389–396.

[3] A. Pfeffer, D. Koller, B. Milch, K. Takusagawa, SPOOK: A system for probabilistic object-oriented knowledge representation, in: Proc. of UAI'99, 1999, pp. 541–550.

[4] D. Koller, A. Pfeffer, Object-oriented Bayesian networks, in: Proc. of AAAI'97, 1997, pp. 302–313.

[5] A. Pfeffer, Probabilistic reasoning for complex systems, Ph.D. thesis, Stanford University (2000).

[6] L. Torti, Structured probabilistic inference in object-oriented Bayesian networks, Ph.D. thesis, Paris VI – UPMC (2012).

[7] M. Jaeger, Relational Bayesian networks, in: Proc. of UAI'97, 1997, pp. 266–273.

[8] K. Kersting, L. D. Raedt, Bayesian logic programs, Technical report no. 151, Institute for Computer Science, University of Freiburg, Germany (April 2001).

[9] L. Getoor, D. Koller, B. Taskar, N. Friedman, Learning probabilistic relational models with structural uncertainty, in: ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries, 2000.

[10] D. Heckerman, C. Meek, D. Koller, Probabilistic models for relational data, Tech. rep., WA: Microsoft Corporation, Redmond (2004).

[11] O. Bangsø, P.-H. Wuillemin, Top-down construction and repetitive structures representation in Bayesian networks, in: Proc. of FLAIRS'00, 2000, pp. 282–286.

[12] O. Bangsø, Object oriented Bayesian networks, Ph.D. thesis, Aalborg University (March 2004).

[13] O. Bangsø, N. Sønderberg-Madsen, F. Jensen, A BN framework for the construction of virtual agents with human-like behaviour, in: Proc. of PGM'06, 2006.

[14] Skoob ANR project (2006–2010).
URL `http://skoob.lip6.fr`

[15] X. Yang, Probabilistic Reasoning in Multi-Agent Systems: A Graphical Models Approach, Cambridge University Press, 2002.

[16] D. Poole, First-order probabilistic inference, in: Proc. of IJCAI'03, 2003, pp. 985–991.

[17] R. de Salvo Braz, E. Amir, D. Roth, Lifted first-order probabilistic inference, in: Proc. of IJCAI'05, 2005, pp. 1319–1325.

[18] R. de Salvo Braz, Lifted first-order probabilistic inference, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (2007).

[19] B. Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, L. P. Kaelbling, Lifted probabilistic inference with counting formulas, in: Proc. of AAAI'08, 2008, pp. 1062–1068.

[20] J. Kisynski, D. Poole, Lifted aggregation in directed first-order probabilistic models, in: Proc. of IJCAI'09, 2009.

[21] L. Torti, P.-H. Wuillemin, Structured probabilistic inference, International Journal of Approximate Reasoning (to appear).

[22] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, B. Taskar, Probabilistic relational models, in: L. Getoor, B. Taskar (Eds.), An Introduction to Statistical Relational Learning, MIT Press, 2007.

[23] L. Torti, P.-H. Wuillemin, C. Gonzales, Reinforcing the object-oriented aspect of probabilistic relational models, in: T. R. Petri Myllymäki, T. Jaakkola (Eds.), Proc. of PGM'10, 2010, pp. 273–280.

[24] N. Friedman, L. Getoor, D. Koller, A. Pfeffer, Learning probabilistic relational models, in: Proc. of IJCAI'99, 1999, pp. 1300–1309.

[25] P.-H. Wuillemin, L. Torti, Structured Probabilistic Inference, International Journal of Approximate Reasoning (2012) 946–968doi:10.1016/j.ijar.2012.04.004.

[26] K. Murphy, Dynamic Bayesian networks: Representation, inference and learning, Ph.D. thesis, University of California (2002).

[27] S. Lauritzen, D. J. Spiegelhalter, Local computations with probabilities on graphical structures and their applications to expert systems, Journal of the Royal Statistical Society 50 (2) (1988) 157 – 224.

[28] G. Shafer, Probabilistic expert systems, Society for Industrial and Applied Mathematics, 1996.

[29] F. Jensen, S. Lauritzen, K. Olesen, Bayesian updating in causal probabilistic networks by local computations, Computational Statistics Quarterly 4 (1990) 269–282.

[30] R. Dechter, Bucket elimination: A unifying framework for reasoning, Artificial Intelligence 113 (1999) 41–85.

[31] A. Madsen, F. Jensen, LAZY propagation: A junction tree inference algorithm based on lazy inference, Artificial Intelligence 113 (1–2) (1999) 203–245.

[32] L. Torti, P.-H. Wuillemin, Structured value elimination with d-separation analysis, in: Proc. of FLAIRS'10, 2010, pp. 122–127.

[33] D. Rose, G. Lueker, R. Tarjan, Algorithmic aspects of vertex elimination on graphs, SIAM J. on Computing 5 (1976) 266–283.

[34] G. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (1990) 393–405.

[35] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.

[36] A. Inokuchi, T. Washio, H. Motoda, A general framework for mining frequent subgraphs from labeled graphs, Fundamenta Informaticae 66 (1–2) (2004) 53–82.

[37] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proc. of ICDM'01, 2001, pp. 313–320.

[38] X. Yan, J. Han, gSpan: Graph-based substructure pattern mining, in: Proc. of ICDM'02, 2002, pp. 721–724.

[39] S. Sanghavi, D. Shah, A. Willsky, Message passing for maximum weight independent set, IEEE Transactions on Information Theory 55 (11) (2009) 4822–4834.

[40] M. Halldórsson, Approximations of weighted independent set and hereditary subset problems, Journal of Graph Algorithms and Applications 4 (1) (2000) 1–16.

[41] S. Cook, The complexity of theorem-proving procedures, in: Proc. of the ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[42] J. Ullmann, An algorithm for subgraph isomorphism, Journal of the ACM 23 (1) (1976) 31–42.

[43] B. McKay, Practical graph isomorphism, Congressus Numerantium 30 (1981) 45–87.

[44] D. Rose, Triangulated graphs and the elimination process, Journal of Mathemactical Analysis and Applications 32 (1970) 597–609.

[45] U. Kjærulff, Triangulation of graphs — algorithms giving small total state space, Tech. Rep. R-90-09, Dept. of Maths and Computer Science, Aalborg University (1990).

[46] J. Flores, J. Gamez, K. Olesen, Incremental compilation of Bayesian networks, in: Proc. of UAI'03, 2003, pp. 233–240.

[47] J. S. Ide, F. G. Cozman, F. T. Ramos, Generating random Bayesian networks with constraints on induced width, in: Proc. of ECAI'04, 2004, pp. 323–327.

[48] M. Garey, D. Johnson, L. Stockmeyer, Some simplified NP-complete problems, in: Proc. of ACM symp. on theory of computing, 1974, pp. 47–63.