

INCREMENTAL COMPILATION OF BAYESIAN NETWORKS BASED ON MAXIMAL PRIME SUBGRAPHS

M. JULIA FLORES* and JOSE A. GÁMEZ†

*Computing Systems Department, University of Castilla-La Mancha,
Campus Universitario s/n, Albacete, 02071, Spain*

**julia.flores@uclm.es*

†*jose.gamez@uclm.es*

KRISTIAN G. OLESEN

*Computer Science Department, Aalborg University,
Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark
kgo@cs.aau.dk*

Received 18 March 2010

Revised 23 December 2010

When a Bayesian network (BN) is modified, for example adding or deleting a node, or changing the probability distributions, we usually will need a total recompilation of the model, despite *feeling* that a partial (re)compilation could have been enough. Especially when considering dynamic models, in which variables are added and removed very frequently, these recompilations are quite resource consuming. But even further, for the task of building a model, which is in many occasions an iterative process, there is a clear lack of flexibility. When we use the term *Incremental Compilation* or IC we refer to the possibility of modifying a network and avoiding a complete recompilation to obtain the new (and different) join tree (JT). The main point we intend to study in this work is JT-based inference in Bayesian networks. Apart from undertaking the triangulation problem itself, we have achieved a great improvement for the compilation in BNs. We do not develop a new architecture for BNs inference, but taking some already existing framework JT-based for probability propagation such as Hugin or Shenoy and Shafer, we have designed a method that can be successfully applied to get better performance, as the experimental evaluation will show.

Keywords: Bayesian networks inference, maximal prime subgraphs, incremental compilation, triangulation, join/junction trees.

1. Introduction

Methods for probabilistic inference have been known for centuries, but it is only within the last two decades that these methods have matured into practical applicable methods. The obstacle has been computational complexity. The joint probability distribution for a set of discrete random variables can be represented by a table, but

the size of such a table is the product of the number of states for each additional variable. Thus it becomes impossible to represent the joint distribution of even moderately-sized domains in practice. In the 1980s Pearl suggested representing domains graphically and exploiting (in)dependence relations between variables in the inference process,³⁴ and his groundbreaking work led to a flourishing interest in the area, resulting in modern tools and techniques for construction and execution of Bayesian networks.

By inference in Bayesian networks we generally refer to the task of obtaining the posterior probabilities for a set of interest variables $X_I \subset \mathcal{V}$ given the evidence e . That is to say, we wish:

$$P(X_i|e) \quad \forall X_i \in X_I \quad (1)$$

There exist other inference tasks such as abductive inference,^{15,33} where the aim is to look for the most likely overall hypothesis or explanation accounting for the current observations.

In both cases, the basic idea in modern techniques is to modularize the domain in question and represent it by a set of tables, so that later procedures for their interaction could be used rather than using one big table. This is achieved by grouping variables according to their dependence relations and these groups of variables are often organized in a tree structure. Variations of this idea were fostered independently by Lauritzen and Spiegelhalter,²⁵ and Shafer and Shenoy,^{35,36} and form the foundation for the Hugin architecture.¹⁹ More recent methods such as lazy propagation,²⁹ penniless propagation,⁵ lazy-penniless propagation,⁶ and methods for stochastic simulation,^{7,30} also take advantage of the modular representation.

The grouping of variables and the organisation of these groups in a tree structure is typically performed offline through a process termed compilation. This process operates with a Bayesian network as input and the resulting structure is called a join tree. The efficiency of the resulting system is proportional to the size of the join tree and therefore considerable effort is devoted to this task. Compilation makes big demands on resources and its complexity increases considerably for large networks. Once the join tree has been constructed future modifications over the network may be necessary. In this case a whole re-compilation is needed. However, changes are normally located in the same area, and if the network is large, a set of changes will typically influence only a small part of the network. For this reason we started working on a compilation that could be carried out partially.

Hence, incremental compilation tries to give an answer to the following question: “*When modifying a Bayesian network, is it absolutely necessary to recompile it from scratch?*”. We could guess that if this modification does not affect the network globally, it might be possible to find a manner to save some time in obtaining the associated tree. In this paper we expose the compilation process itself to modularization. We analyze the procedures and split the problem into subproblems, thereby enabling online compilation or at least more efficient offline procedures. Fig. 1 tries to illustrate this simple idea: let us assume that we have an initial

Bayesian network BN and its associated join tree \mathcal{T} , obtained by a compilation process. Later, at a certain moment, a set of modifications have to be carried out on this initial network, which will be transformed into a different network BN' . For this new network \mathcal{T} will clearly not be a valid join tree, but there must be another one, \mathcal{T}' , which is. The key to incremental compilation would then be to find a way (obviously, quicker than a complete recompilation of BN') of obtaining, from this new network, the/a new tree that corresponds to it. Thus, the question we previously posed could be reformulated as follows: “*If we know the modifications performed on our network, and starting from the initial join tree corresponding to the initial network (\mathcal{T}), could we obtain the new join tree (\mathcal{T}')?*”. This is precisely the question represented by “?” in Fig. 1. And the answer is our concrete method for carrying out this incremental compilation process.

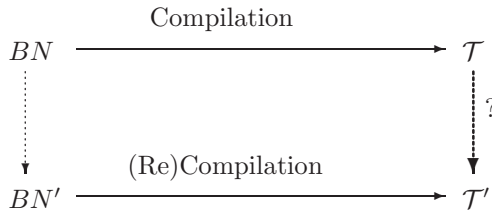


Fig. 1. Scheme that represents the general idea for our approach for the incremental compilation of a Bayesian network BN .

This issue has hardly been studied in BNs literature. In 1999, Olesen and Madsen present an algorithm to obtain the so-called Maximum Prime subgraph Decomposition (MPD) of a Bayesian network by using the join tree as the base structure.³² In that paper they enumerate a number of possible applications of their algorithm, one of these being the study and development of MPD-based incremental compilation. This was the seminal idea for our preliminary work on this topic where a procedure for MPD-based Incremental Compilation of BNs was proposed.¹⁴

There is however some references in the literature that must be taken into account. Before our preliminary work,¹⁴ little attention had been directed towards incremental compilation in the literature: Darwiche considers dynamic generation of join trees,⁸ but his aim is to produce efficient elimination orders for specific queries posed to the model rather than modifications over the network. His method is focused on efficient processing by online generation of specific join trees and it does not take modifications of the structure of the underlying network into account; A different study is carried out by Draper,⁹ whose main goal is to show how to build join trees avoiding triangulation, or at least without explicitly considering it. To do this, a family graph is initially computed and then transformed into a join tree by using a set of heuristics. This approach can also be applied in case of dynamic changes in the network structure where the same set of heuristics are applied to dynamically maintain the join tree. A disadvantage of the method is that it is

difficult to identify the relevant set of heuristics, and the resulting join trees are often suboptimal.

Later in time Acar *et al.*^{1,37} approached the problem of dealing with dynamic changes in the graph, but taking factor graphs as original model instead of BNs. They devise a procedure to accommodate dynamic changes over the current cluster tree instead of constructing it from scratch from the modified factor graph. On the other hand, graph decomposability has been also exploited in other tasks different from inference, as it is the case of machine learning. Thus, Xiang and Lee propose its use when learning undirected structures from data, in particular Decomposable Markov Networks.⁴¹ The idea is to identify the subgraphs involved in an incremental operation, e.g. adding a link, and restrict score computations only to those subgraphs

Currently, there exist several studies which use our preliminarily proposed MPD-based Incremental Compilation technique or which cite it as an important approach to the modularity problem when compiling or learning BNs.^{3,16,17,22} In this paper we present an archival version of MPD-based incremental compilation,¹⁴ but completing it with an in-depth experimental evaluation.

In the following sections, we will first introduce the basic definitions, processes and associated notation (Sec. 2). In Sec. 3 we present Maximal Prime Subgraph Decomposition, which will be necessary for the Incremental Compilation method described in Sec. 4. The experimentation carried out for this study is explained in Sec. 5, where the corresponding results are also analyzed. Finally we present the general conclusions in Sec. 6.

2. Preliminaries

2.1. Notation and basic definitions

Definition 1. Bayesian Network. *A Bayesian network is a formalism used in probabilistic reasoning with two components:*

- The **qualitative** side defined as a directed acyclic graph $G = (V, E)$, V being a set of variables and E a set of directed edges.
- And the **quantitative** side given by a probability distribution P over the variables in V , where for every variable or node in the graph there is a conditional probability distribution $P(X_i | pa(X_i))$, $pa(X)$ being the parents of X , that is those Z_j s.t. $Z_j \rightarrow X \in E$.

The Bayesian network is a nice representation to constitute the Knowledge Base for a probabilistic expert system, thanks to its graphical visualisation. However, inference is usually carried out over a secondary structure free of (undirected) cycles. One of the most extended techniques for inference in Bayesian networks is the use of a tree of cliques (see Defs. from 2 to 5) known as **join tree** or junction tree.²⁰

Definition 2. Subgraph. *Let $G' = (V', E')$ be a subgraph of $G = (V, E)$ when*

$V' \subseteq V$ and E' contains all edges in E between two nodes X_i and X_j as long as $X_i \in V'$ and $X_j \in V'$. This subgraph can also be denoted as $G^{\downarrow V'}$.

Definition 3. Complete undirected graph. Let $G = (V, E)$ be a complete graph iff for every pair $X_i, X_j \in V$, $X_i - X_j \in E$.

Definition 4. Clique. Let G be an undirected graph, then all the maximal complete subgraphs in G are called cliques.

Definition 5. Join Tree. Let G be the set of cliques from an undirected graph, and let the cliques of G be organised in a tree \mathcal{T} . Then \mathcal{T} is a join tree if for any pair of clique nodes C_i, C_j all nodes on the path between C_i and C_j contain the intersection $C_i \cap C_j$.

As mentioned above a BN is represented by a directed acyclic graph whereas the join tree is constructed from an undirected graph. Note that even if this structure has been broadly used and referenced in BNs research, this concept had already been broadly and previously used in graph literature.² This is because the transformation from a BN to a corresponding and valid \mathcal{T} needs a transformation process called **compilation** where the original graph G is first moralised, giving rise to an undirected graph that will subsequently be triangulated (Fig. 2).

Definition 6. Moral graph. Given a directed acyclic graph G , its corresponding moral graph G^M is formed by connecting nodes that have a common child, and then making all edges in the graph undirected.

Definition 7. Triangulated graph. An undirected graph G is said to be triangulated or chordal, G_T , if any cycle of length greater than 3 has a chord.

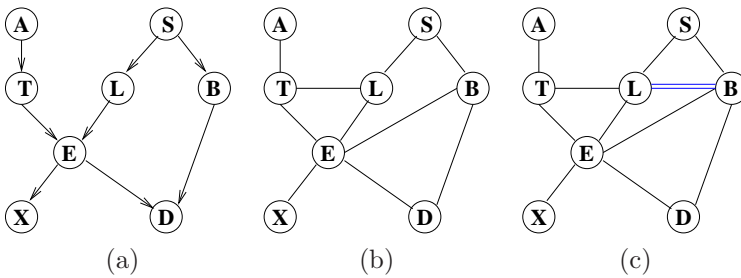


Fig. 2. (a) The Asia network. (b) Moral graph for Asia. (c) Triangulated graph for Asia.

Therefore, the moral graph can be triangulated by adding new edges (called *fill-ins*) until the necessary condition in Def. 7 holds. **Triangulation** is a key step, since efficiency of probability propagation heavily depends on the quality of the obtained triangulation, but finding the optimal triangulation is an NP-hard problem.⁴⁰ It is quite usual to triangulate a graph using an elimination sequence σ or ordering that contains all variables, this sequence being the input of a procedure

able to provide the added fill-ins.¹³ This is why triangulation is sometimes considered as the task of finding the best elimination sequence, but the search space of all possible permutations yields $|V|!$ possibilities. In this study, we will use the set of added links, or fill-ins, \mathcal{F} . Finally, we present the concept of minimal triangulation (see Def. 8), as it will be necessary in Sec. 3.

As an example, in Fig. 2(a) the graph representing the Asia network is depicted,²⁵ leading to the moral graph in Fig. 2(b) and a possible triangulation is shown in Fig. 2(c) where the added fill-in ($\mathcal{F} = \{L-B\}$) is marked with a double line.

Definition 8. Minimal Triangulation. Let \mathcal{F} be the set of fill-ins added during triangulation, for an undirected graph G , i.e. $G_T = (V, E \cup \mathcal{F})$, \mathcal{F} is said to be *minimal* if $\nexists \mathcal{F}'$ such that $\mathcal{F}' \subset \mathcal{F}$ and \mathcal{F}' is a valid triangulation for G . That is, if we remove any edge from \mathcal{F} , the resulting set \mathcal{F}' is no longer a valid triangulation for G .

Kjærulff proposed an efficient method called *recursive thinning* to transform any triangulation containing redundant fill-ins into a minimal one.²³

2.2. Compilation process

Most popular knowledge engineering tools for construction and execution of probabilistic expert systems based on Bayesian networks (BNs), such as for example HUGIN or NETICA,^{18,31} work with two representations: (1) A direct representation of the BN as illustrated by the edit window in Fig. 3 and (2) A computational structure known as the join tree (JT), illustrated by the run window in Fig. 3.

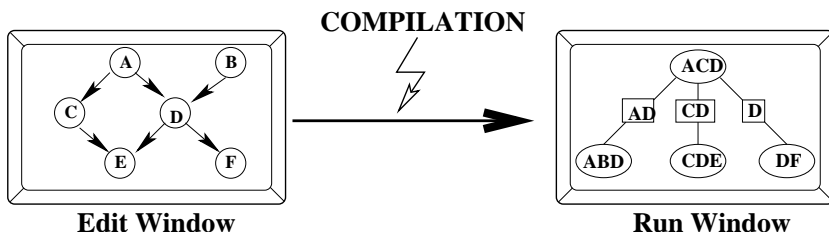


Fig. 3. Two representations of a Bayesian network.

A BN is constructed or modified in the first representation and inference is performed over the second one. Each time the knowledge engineer switches from editing to execution, a *compilation* builds the join tree from scratch. If the network is large, then the CPU time required to perform a new compilation will be considerable. Therefore, it is desirable that, once a join tree representation of the Bayesian network has been generated, incremental changes in the network should produce an update of the previous join tree, and not a new full compilation process. Apart

from the *efficiency* reason, there could be more reasons for preferring incremental compilation to total compilation. Draper emphasises *stability* of the join tree as a desirable property of incremental compilation.⁹ It is anticipated that a considerable effort will be made to produce an efficient join tree and that incremental changes to an existing (near) optimal join tree will produce more stable results.

The method we propose for incremental compilation identifies the parts of the join tree that are affected by changes in the BN, and reconstructs only these parts. It also glues the new substructures into the original join tree instead of the outdated parts. This approach ensures a stable and, most of the time, efficient resulting join tree. The method builds on a third representation of the underlying BN, the *Maximal Prime subgraph Decomposition (MPD) tree*.³²

3. Maximal Prime Subgraph Decomposition (MPD)

Let us now formalise the concept of maximal prime subgraph, and the needed preliminary definitions.

Definition 9. Graph decomposition. *Let $G = (V, E)$ be an undirected graph, and let A and B be two sets of vertices in G , G can be decomposed into A and B if and only if the following conditions are satisfied:*

- $A \cup B = V$,
- $A \setminus B \neq \emptyset$,
- $B \setminus A \neq \emptyset$,
- $A \setminus B$ and $B \setminus A$ are separated by $A \cap B$ and
- $A \cap B$ is a complete subset (called **clique separator**).

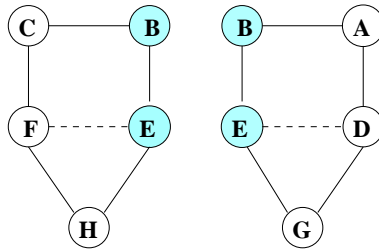


Fig. 4. Graph decomposition into $\{C, F, H, B, E\}$ and $\{B, E, A, D, G\}$.

Definition 10. Decomposable graph. *If a graph G and its subgraphs can be decomposed recursively until all the subgraphs are complete, then the graph is decomposable.*

Notice that a graph could be decomposed without being decomposable, as in Fig. 4. The concepts of triangulation and decomposition can easily be related from

previously presented ideas, as triangulated graphs are guaranteed to be decomposable and that is to a certain extent the justification for the necessity for a triangulation step in compilation.²⁶ This is the reason why from here on, we will refer to a decomposable graph as a triangulated graph.

A graph is then said to be *reducible* if it can be decomposed, that is, its set of nodes contains a clique separator, otherwise the graph is said to be *irreducible/prime/non-separable*. And this leads directly to the following concepts:

Definition 11. Maximal Prime Subgraph (MPS). A subgraph $G(A) = (V, E)^{\downarrow A}$ of a graph G is a Maximal Prime Subgraph of G if $G(A)$ is irreducible and $G(B)$ is reducible $\forall B$ such that $A \subset B \subseteq V$.

Definition 12. Maximal Prime Subgraph Decomposition (MPD). Let $G = (V, E)$ be an undirected graph. Its Maximal Prime Subgraph Decomposition is the set of induced maximal prime subgraphs of G resulting from a recursive decomposition of G .

It can be proved that this decomposition is unique for an undirected graph,³² as it is the moral graph. In Fig. 5(c) we see how the Asia network is divided into five Maximal Prime Subgraphs, which can also be structured into a tree (Fig. 5(b)).

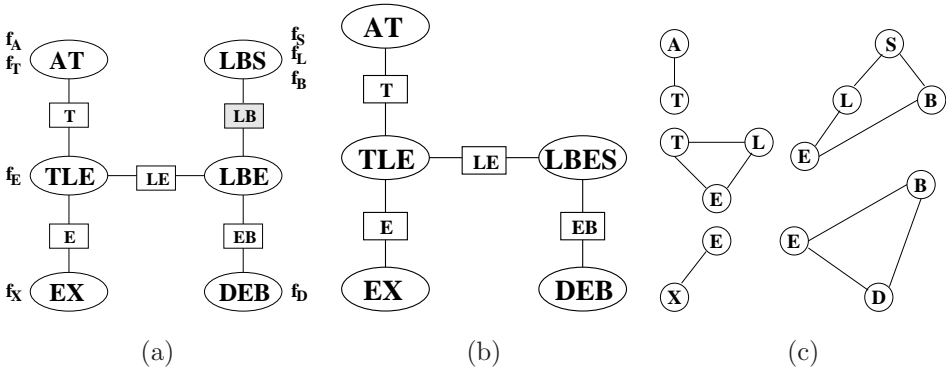


Fig. 5. (a) A JT for Asia. (b) MPD tree for Asia. (c) MPD for Asia.

When the decomposition obtains a set of solvable subgraphs, this is a suitable tool for *divide and conquer* algorithms, which provide a global solution as the sum of local solutions for smaller and independent graphs. In particular, we use the MPD^a of an undirected graph as an intermediate step in our new approach for triangulation. This idea consists of working separately on different parts of the initial graph.³² In our case, the task to do separately will be the triangulation for each subgraph.

^aAlso known as decomposition by clique separators.

The decomposition of the graph of a Bayesian network into its maximal prime subgraphs is integrated into the well known procedure for construction of join trees for Bayesian networks. We know that the maximal prime subgraphs of G^M are formed by aggregating adjacent cliques of \mathcal{T}_{min} ^b connected by a separator which is incomplete in G^M . CONSTRUCT_MPD_TREE (Alg. 1^c) performs this process and returns a join tree \mathcal{T}_{MPD} , where the nodes represents the maximal prime subgraphs.

Algorithm 1 Obtain the MPD Tree or \mathcal{T}_{MPD} from a JT.

```

1: function CONSTRUCT_MPD_TREE(Join Tree  $\mathcal{T}_{min}, G_M$ )
  ▷ Join Tree  $\mathcal{T}_{min}$  must have been obtained by a minimal triangulation.
  ▷  $G_M$  is the corresponding moral graph to the network.
2:    $\mathcal{T}' \leftarrow \mathcal{T}_{min}$ 
3:   repeat
4:     for all Separator  $S \in \mathcal{T}'$ ,  $S$  connects clusters  $C_1$  and  $C_2$  do
5:       if  $\neg(\text{COMPLETE\_GRAPH}(G_M^{\downarrow S}))$  then
6:         AGGREGATE( $C_1, C_2, \mathcal{T}'$ )
7:       end if
8:     end for
9:   until  $\forall S_k \in \mathcal{T}'$ ,  $S_k$  is complete
  ▷ Until All separators  $S$  in  $\mathcal{T}'$  are complete for  $G_M$ .
10:  return  $\mathcal{T}'$ 
11: end function

```

The resulting join tree for Asia network (see Fig. 2) is shown in Fig. 5 (a) and a check of the separators in the moral graph yields the maximal prime subgraph decomposition tree shown in part (b). Part (c) gives the MPSs of the Asia network.

The structure of the join tree is a refinement of the MPD tree (although constructed in the opposite order), where a node in the MPD tree may be expanded into one or more cliques in the join tree. It is this structural correspondence that is exploited in our method for incremental compilation.

Although other methods have been proposed to obtain the MPD of an undirected graph,^{27,38,39} the one presented by Olesen and Madsen is especially interesting for us, since it is based on the join tree constructed from a BN.³² The decomposition of the graph into MPSs is returned by the form of a tree, which we will call MPD Tree, sometimes denoted as \mathcal{T}_{MPD} .

^bJoin tree constructed from a minimal triangulation.

^cIn algorithms we use symbol ▷ to indicate comments.

The process for obtaining this tree is indicated in Alg. 2, OBTAIN_MPD_TREE:

Algorithm 2 Obtain the MPD Tree or \mathcal{T}_{MPD} from a graph.

```

1: function OBTAIN_MPD_TREE(Graph  $G$ )
  ▷ We assume this graph  $G$  is the graphical part for a Bayesian network  $BN = (G, P)$ .
2:    $G_M \leftarrow$  MORALISE_GRAPH( $G$ )
3:   Triangulation  $T \leftarrow$  TRIANGULATE_GRAPH( $G$ )
  ▷  $G_M^T$  will be the triangulated moral graph.
4:    $T_{min} \leftarrow$  MAKE_MINIMAL_TRIANGULATION( $T$ )
  ▷ In our case we will use the algorithm called recursive thinning.
5:    $\mathcal{T}_{min} \leftarrow$  CONSTRUCT_JOIN_TREE( $G_M^{T_{min}}$ )
  ▷ This can be done for instance by means of the algorithms shown in11.
6:   return CONSTRUCT_MPD_TREE( $\mathcal{T}_{min}, G_M$ )
  ▷ Alg. 1 shows this process.
7: end function

```

Basically, if we guarantee that in the compilation process the JT \mathcal{T} has been obtained from a minimal triangulation, the corresponding \mathcal{T}_{MPD} can easily be found by aggregating those separators which are not complete subgraphs in the moral graph, as in Fig. 5(a), where separator (LB) is shaded because in 5(a) this set of nodes does not yield a complete subgraph. Then, cliques (LBE) and (LBS) are joined together into one Maximal Prime Subgraph ($LBES$) as shown in Fig. 5(b).

Since the tree of MPSs can be seen as an intermediate structure that is located in between the moral graph and the triangulated graph, this feature can be useful for both theoretical algorithms and implementation (data structures).

Two different (minimal) triangulations produce two different join trees, and those trees could also differ in tree size.^d Notice that in terms of tree topology the same triangulation could also lead to distinct trees, although their total space size will be exactly the same, since they present the same set of cliques. For example, in Fig. 5(a), the clique (EX) could be connected either to (LBE) or (DEB) (instead of (TLE)), thus producing other valid JTs that are topologically different from the one depicted in the figure. This will also have an impact on the topology of the MPD Tree obtained, since it is based on the initial JT. However, it should be noted that this will not affect the decomposition because fill-ins are not considered in the completeness of a separator, only the moral graph is now considered.

Figure 5 shows the construction (fusion of cliques into MPSs) for the join tree in Fig. 5(a) regarding the moral graph in Fig. 2(b). Those cliques that can be fused are (LBS) and (LBE), since the shaded separator is not complete in the moral graph (Fig. 2(b)). Notice that $[T], [E]$ and $[EB]$ induce completely connected subgraphs in the moral graph, so they did not need any merging to form a maximal prime subgraph.

^dSummation of the space state size (i.e. probability table size) of every clique.

Among MPD properties we emphasize this: “If C is a clique of \mathcal{T}_{min} and all separators connected to C are complete in G_M , then C is a clique of G_M .”³² This fact prevents any node in the tree subgraphs from producing fill-ins connecting nodes not in this same subgraph.^e

4. MPD-Based Incremental Compilation

4.1. The role of MPD within incremental compilation

The feasibility of reusing an already existing JT in order to obtain the new one while considering only the modifications in the network had not really been studied thoroughly before. In order to perform incremental compilation we propose the recompilation of only those parts of the JT which may have been affected by the network’s modifications. To do so, we exploit MPD in determining the minimal set of subgraphs that have to be recompiled, and thereby the minimal subtree(s) of the JT that should be replaced by new subtree(s). Changes are normally located in the same area, and if the network is large, a set of changes will typically influence only a small part of the network. These two reasons encouraged us to work on a compilation that could be carried out partially.

If we guarantee that the triangulation to construct \mathcal{T} is minimal, this \mathcal{T} leads us to a valid \mathcal{T}_{MPD} following the steps in Fig. 6.

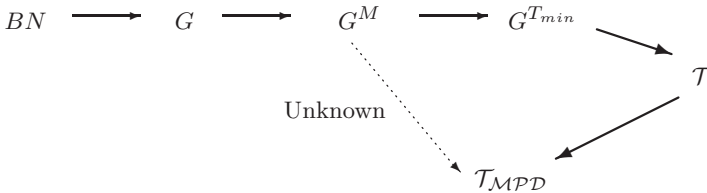


Fig. 6. Graphical process that indicates how to reach the \mathcal{T}_{MPD} from a Bayesian network $BN = (G, \mathcal{P})$, using as an intermediate step the join tree \mathcal{T} .

As Fig. 1 shows, we were searching for a way to go from \mathcal{T} to the new tree \mathcal{T}' , without requiring a full recompilation. And the MPD Tree \mathcal{T}_{MPD} can be understood as an intermediate structure between the BN and the tree \mathcal{T} , also bearing in mind that (see Fig. 6) from this one (\mathcal{T}) \mathcal{T}_{MPD} can easily be constructed.

In Fig. 7, question marks refer to the two steps for which the execution is not initially clear, while the rest has been described above (see Fig. 6). It is precisely these two unknown points that our proposed algorithm will give an answer for.

^eThis is the underlying idea of requiring a minimal triangulation for \mathcal{T} .

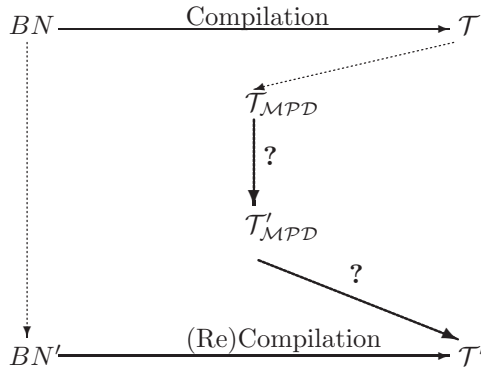


Fig. 7. Diagram that represents an overview of our Incremental Compilation process for a Bayesian network BN by means of the MPD.

4.2. Possible modifications to be considered

The set of changes in an existing BN range from simple modifications, e.g. adjustments to numerical parameters in the (conditional) distributions for variables, to complex structural reorganisation of variables and their links, possibly altering large parts of the BN. Below we will briefly describe the possible modifications we will consider. There are two outstanding properties: first, we aim towards the generation of a general (and efficient) method that could process all kinds of situations; and second, we also want a method capable of dealing with more than one modification at a time, that is, capable of processing a group of changes as well as single ones, according to user preferences.

4.2.1. Modification of potentials

The simplest modification of a BN is altering the (conditional) probability distribution for a variable. Such a change is purely quantitative, and it is straightforward as the structure of the join tree will remain unchanged. In this case we simply replace the current table with the modified one.

4.2.2. Modification of the states of a variable

Changes in the state space of a variable can alter the structure of the optimal join tree, because the triangulation typically takes the state space of variables into account. Notice that these state variations will not have an effect when treewidth (i.e. the size of a largest clique in a junction tree minus one) is considered. A full treatment of incremental compilation should, of course, consider this class of changes, but the overall efficiency of the resulting computational structure remains roughly the same. We shall therefore disregard the structural implications of such changes from further considerations in the present treatment of the subject. What remains is then the modified structure of the potentials of the altered variable and

its children and this is again taken care of by replacing old potentials with new ones for the affected variables.

4.2.3. *Modifying the graph structure*

Removing an arc: Removal of an arc in a BN can be a straightforward change. If, for example, two nodes without parents share a child which is not otherwise connected, removal of an arc between them will not lead to changes in the moral graph, as the link would appear here anyway due to moralisation. At the other extreme the removal of an arc could break several loops in the BN and a considerably simpler join tree could result from a retriangulation of the network. In such cases it is beneficial to be able to identify a minimal part of the join tree that could be affected by the change and concentrate on a retriangulation of only that part.

Adding an arc: As with removal of an arc, the addition of a new one is sometimes straightforward. A simple situation is symmetric to the simple case above, where two nodes without parents share a child which is not otherwise connected. Addition of an arc between them will not lead to changes in the moral graph, as the link would appear here anyway due to moralisation. At the other extreme, the addition of an arc could create several cycles in the BN and in this case large parts of the join tree could be affected. Sometimes a complete retriangulation of the network is required, and again it is beneficial to be able to identify the minimal part of the join tree that could be affected by the change and concentrate on a retriangulation of only that part.

Removing a node: The removal of a node from the BN will include removal of all arcs connected to that node. If all arcs are removed first, the removal of the node is simple. If not connected to any other node, the node will constitute an island in the BN and consequently it can simply be deleted. We should point out here that we refer to the removal of a node graphically (edit mode) from the network. This has nothing to do with other probabilistic techniques such as the removal of a node by marginalisation.

Adding a node: The addition of a new node is similarly simple, if we connect it to the BN afterwards, arc by arc. The node is simply added to the BN and the procedure for adding arcs is called when it is linked with the existing BN.

This study is mainly focused on structural changes and this is the reason for not considering modifications of type 1 and 2, since they do not affect the JT structure. We shall therefore concentrate on the addition and deletion of nodes and arcs.

4.3. *MPD as a tool for incremental compilation*

The problem we investigate is summarised in Fig. 1, and partially answered in Fig. 7. As mentioned above, modifications of the BN can result in everything from

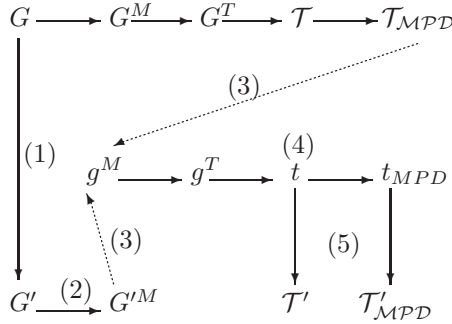


Fig. 8. Integrated overview of the MPD-based IC process. g and t refer to partial (affected) graph and join tree.

trivial to very complex modifications of the join tree. We are therefore looking for ways to limit the parts of the join tree that have to be recompiled.

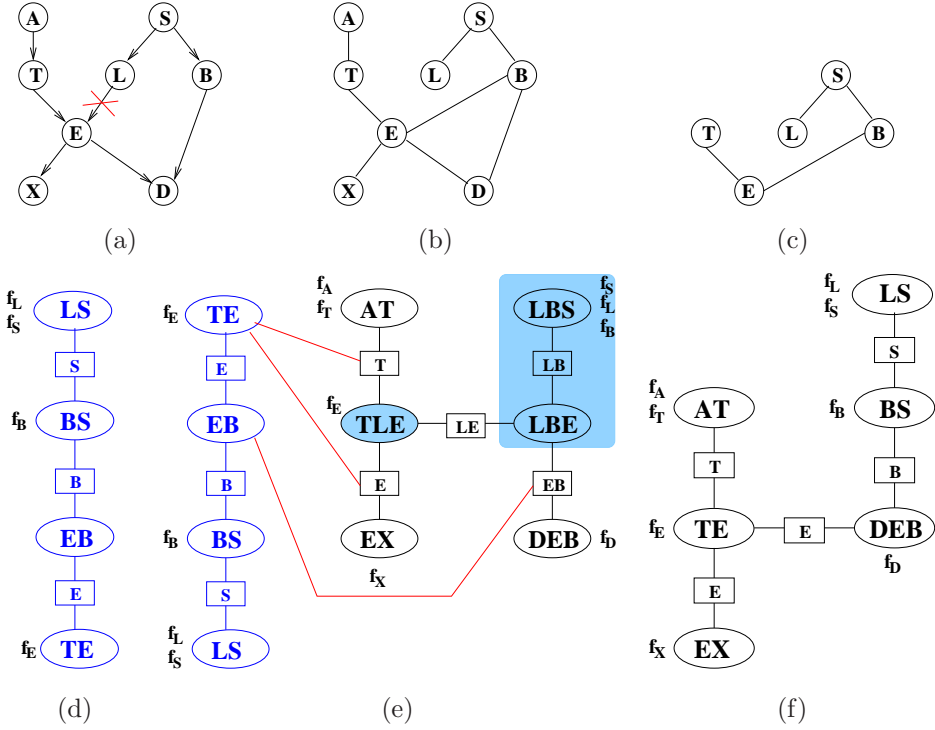
It is known that the maximal prime subgraphs can be triangulated independently.^{12,32} Then, to attain our goals we can proceed as follows: Each time a BN is recompiled we identify the set of MPSs affected by the modifications since the last compilation, and only these MPSs will be re-triangulated. Our expectation is that only a few subgraphs of \mathcal{T}_{MPD} will be influenced by the modifications, and consequently only a small part of the graph will have to be re-triangulated. Thus, the major part of the join tree remains unmodified and can be reused.

Let us suppose that the process starts with a user making modifications in the edit window of Fig. 3. The result of this editing process is an updated version of the BN (G') (step (1) in Fig. 8). A list of the modifications is constructed during this step. This list serves as input to INCREMENTALCOMPILATION algorithm, that is activated when the user decides to obtain a new join tree, that is, formally speaking, when moving to the inference window.

Once the user decides to produce a join tree for the new network G' , the first step of incremental compilation is to modify the moral graph (Step (2) in Fig. 8). Notice that the moral graph plays a crucial role in \mathcal{T}_{MPD} construction. We will pay special attention to this step in MODIFYMORALGRAPH (Alg. 4).

As an example, let us suppose that in the network Asia (Fig. 2(a)) the user has removed link $L \rightarrow E$ and has invoked the IC algorithm. Then, Fig. 9 shows the modified BN and its corresponding moral graph in parts (a) and (b) respectively. Note that the moral link T-L has also been removed.

Now, we proceed to step (3) in Fig. 8. This is the key point in our algorithm, where we identify the minimal set of MPSs which may be affected by the modifications performed over the BN. Below we shall give details of this for the different modifications, but for the moment, let us assume the existence of an algorithm which marks the MPSs in \mathcal{T}_{MPD} affected by a modification. This algorithm marks MPSs (TLE) and ($LBES$). In the example, there is only one connected marked subtree (TLE) – [LE] – ($LBES$), but in general, when all modifications have been


 Fig. 9. MPD-based IC of Asia when removing $L \rightarrow E$.

processed by the marking algorithm, there may be more connected parts of \mathcal{T}_{MPD} that have been marked. As an example, consider the scenario in which the marking algorithm marks MPSs: (AT) , (TLE) and (DEB) ; which gives rise to two marked connected subtrees: $(AT) - [T] - (TLE)$ and (DEB) . For each of these marked connected subtrees we proceed, in turn, with the following steps.

Let g^M be the subgraph of G^M induced by the set of variables included in a connected marked subtree of \mathcal{T}_{MPD} . Figure 9(c) shows g^M as the projection of G^M over $\{T, L, E, B, S\}$. In step (4) in Fig. 8 we obtain a JT and a MPD tree for g^M . Join tree t is available by OBTAIN_MPDTREE (Alg. 2) but avoiding step in line 1, since the graph is already moralised and the corresponding MPD tree t_{MPD} by CONSTRUCT_MPDTREE (Alg. 1). In the example, Fig. 9(d) shows both structures, which is the same in this case since every MPS corresponds uniquely to one clique.

Finally, during step (5) in Fig. 8 both \mathcal{T} and \mathcal{T}_{MPD} are updated by using the newly obtained structures t and t_{MPD} . The process is completely analogous in both cases and it only differs in the tree to which it is applied. For each separator S connecting a marked subtree with an unmarked cluster we reconnect S to a cluster of t (t_{MPD}) having maximal intersection with S . Upon completion, the marked clusters (and separators between them) are deleted. In the example, the

separators connecting the marked connected subtree with the rest of \mathcal{T}_{MPD} are $[T]$, $[E]$ and $[EB]$. These separators are connected with the new graphical structure as shown in part (e) of Fig. 9. Finally, part (f) of the same figure shows the final structure, obtained after removing the outdated part of the tree (the marked MPSs). CONNECT (Alg. 5) details this procedure.

4.4. Incremental compilation algorithm

INCREMENTALCOMPILATION (Alg. 3) details the process described above:

Algorithm 3 Performs IC for a list of modifications in the BN.

```

1: procedure INCREMENTALCOMPILATION (Modification list ModList)
2:   for all Modification mod  $\in$  ModList do
3:      $L \leftarrow$  MODIFYMORALGRAPH(mod)
4:     switch mod do
5:       Case Add node X: ADDNODE(X)
6:       Case Delete node X: REMOVENODE(X,  $M_X$ , nil)
7:       Case Delete link  $X \rightarrow Y$ : REMOVELINK( $L$ ,  $M_Y$ , nil)
8:       Case Add link  $X \rightarrow Y$ : ADDLINK( $L$ )
9:     end switch
10:  end for
11:  for all Connected marked subtree  $T_{MPD} \in \mathcal{T}_{MPD}$  do
12:     $T \leftarrow \mathcal{T}$  corresponding to  $T_{MPD}$ 
13:    for all Clique  $C_i \in T$  do MARKCLIQUE( $C_i$ )
14:    end for
15:     $C \leftarrow$  any cluster of  $T$ 
16:     $M \leftarrow$  any cluster of  $T_{MPD}$ 
17:     $V \leftarrow$  {all variables included in  $T_{MPD}$ }
18:     $t \leftarrow$  CONSTRUCTJOINTREE( $g^M$ )
     $\triangleright g^M$  is the projection of the current graph  $G^M$  over the set of variables included in
     $T_{MPD}$ .
19:     $t_{MPD} \leftarrow$  AGGREGATECLIQUES( $t$ )
20:     $\mathcal{T} \leftarrow$  CONNECT( $t$ ,  $C$ , nil)
21:     $\mathcal{T}_{MPD} \leftarrow$  CONNECT( $t_{MPD}$ ,  $M$ , nil)
22:    DELETE( $T$ )
23:    DELETE( $T_{MPD}$ )
24:  end for
25: end procedure

```

In order to simplify the header of the algorithms presented in this section, we suppose that the main graphical structures are accessible, that is, we will refer to G^M , \mathcal{T} and \mathcal{T}_{MPD} without the need of passing them as a parameter to the algorithms. Before we proceed we need some notation. The potential of X is assigned to a specific clique in \mathcal{T} , which contains the family of X . In the following, we will use C_X to identify this clique and, likewise, M_X will identify the MPS in \mathcal{T}_{MPD} which has the family of X associated. In figures this will be indicated with a f_X next to the corresponding clique/MPS.

The first loop of the algorithm iterates over all modifications. For each modification we adjust the moral graph by `MODIFYMORALGRAPH`, Alg. 4, which maintains a list of all links that are affected by the current modification. This is relevant for addition and deletion of arcs, and `MODIFYMORALGRAPH` returns a list, L , with the added (deleted) link and with the induced added (deleted) moral links.

Algorithm 4 Performs the corresponding modifications to the moral graph implied by the *mod* which is being processed.

```

1: function MODIFYMORALGRAPH(Modification mod)
  ▷ This function will also return the set of links relevant for the modification if that
  applies.
2:    $L \leftarrow \emptyset$ 
  ▷  $L$  will be a LinkList containing the relevant links in this operation.
3:   switch mod do
4:     Case Add node  $X$ : add a new (isolated) node  $X$  to  $G^M$ 
5:     Case Delete node  $X$ : remove  $X$  from  $G^M$ 
  ▷ We assume that we are going to delete a disconnected node, if it presents edges they
  will be removed by means of modification Delete link first.
6:     Case Add link  $X \rightarrow Y$ : add  $X \rightarrow Y$  to  $L$  together with all
7:     new links needed to make  $Y \cup \text{parents}(Y)$  a complete sub-graph.
8:     Case Delete link  $X \rightarrow Y$ :
9:     if ( $\text{children}(X) \cap \text{children}(Y) = \emptyset$ ) then
10:      delete  $(X, Y)$  from  $G^M$ 
11:      add  $(X, Y)$  to  $L$ 
12:     end if
13:     for all  $Z_i \in \text{parents}(Y) \setminus \{X\}$  do
14:       if ( $(\text{children}(Z_i) \cap \text{children}(X) = \{Y\})$  and  $(Z_i \rightarrow X$ 
15:       or  $X \rightarrow Z_i)$  not in  $G$ ) then
16:         delete  $(X, Z_i)$  from  $G^M$ 
17:         add  $(X, Z_i)$  to  $L$ 
18:       end if
19:     end for
20:     end switch
21:   return  $L$ 
22: end function

```

The list, L , returned by algorithm `MODIFYMORALGRAPH` is passed on as argument to the relevant procedure, which marks affected Subgraphs in \mathcal{T}_{MPD} . This result of steps 1-10 in `INCREMENTALCOMPILATION` is an MPD-tree with (possibly several) connected marked subtrees. In the second part of `INCREMENTALCOMPILATION` (lines 11-24) we iterate over these subtrees and adjust \mathcal{T} and \mathcal{T}_{MPD} by `CONNECT` (Alg. 5). The pattern for this algorithm may not be immediately transparent. The recursive control structure acts on the two last parameters where the former (the second parameter) is the cluster to which the algorithm is applied, and the latter (the third parameter) is the caller. The structure traverses a marked subtree, avoiding loops by a check that the caller is not re-visited. This pattern will be also used in procedures for removing arcs and links.

Algorithm 5 Performs connection between the new *partial* (JT or MPD) tree to the old remaining part.

```

1: procedure CONNECT(Cluster_Tree  $t$ , Cluster  $C_i$ , Cluster  $C_j$ )
2:   for all Separator  $S$  between  $C_i$  and  $C_k \neq C_j$  do
3:     if  $C_k$  is unmarked then
4:       locate cluster  $C \in t$  such that  $C \cap C_k$  is maximal
5:       Connect  $C$  with  $C_k$  by  $S$ 
6:       if  $S == C$  then amalgamate  $C$  and  $C_k$ 
7:       end if
8:     else CONNECT( $t$ ,  $C_k$ ,  $C_i$ )
9:     end if
10:  end for
11: end procedure

```

We shall now go through the details of the marking of MPSs, that is, the procedures called from the first loop of INCREMENTALCOMPILATION. It is silently assumed that whenever an MPS is marked, the corresponding cliques in the join tree will also be marked.

4.5. Removing a link

REMOVELINK (Alg. 6) marks the MPSs affected by the removal of $X \rightarrow Y$.

Algorithm 6 Marks subgraphs when deleting a link in the IC process.

```

1: procedure REMOVELINK(LinkList  $L$ , MPS  $M_Y$ , MPS  $M_Z$ )
2:   Mark  $M_Y$   $\triangleright M_Y$  is the MPS containing family of variable  $Y$ 
3:   for all Neighbour  $M_K \neq M_Z$  of  $M_Y$  do
4:      $S \leftarrow$  separator between  $M_Y$  and  $M_K$ 
5:     if  $L \cap \text{links}(S) \neq \emptyset$  then
6:       REMOVELINK( $L$ ,  $M_K$ ,  $M_Y$ )
7:     end if
8:   end for
9: end procedure

```

Parameter L is the list of (induced) moral links (returned by MODIFYMORALGRAPH). Notice, that when the algorithm is called the first time (from INCREMENTALCOMPILATION) then $M_Z = \text{nil}$.

Let us suppose that the link $X \rightarrow Y$ has been deleted from G . Then M_Y has been affected and we have to investigate if more MPSs have to be re-triangulated due to a side effect of the deletion of $X \rightarrow Y$. Therefore, we should include the neighbours of M_Y in the set of MPSs to re-triangulate, only if the separator between them is no longer complete. To do this, we look to see if the disappearance of the link $X - Y$ or of any other induced link $Z - X$ causes some separator to become incomplete in G^M . Of course, if a new MPS is marked because of this search, then we have to verify the same condition among their neighbours and so on.

As a case of study, let us go back to the example used during the overview of our IC method, that is, the removing of link $L \rightarrow E$ from the Asia network (Fig. 9). In this case, the parameters received by REMOVELINK are $L = \{(L, E), (T, L)\}$, $M_Y = (TLE)$ and $M_Z = nil$. Therefore, MPS (TLE) is marked in step 2. Of the three separators connected to (TLE) , only $[LE]$ will be considered, because the other two contain only one variable. As $[LE]$ contains one of the removed links, the MPS $(LBES)$ connected to (TLE) by this separator, is also marked by a recursive call of REMOVELINK. Therefore, in this case, the subtree $(TLE) - [LE] - (LBES)$ is marked by the algorithm.

4.6. Removing a node

REMOVENODE (Alg. 7) marks all MPSs containing X , and also deletes X from all MPSs and separators containing it in order to obtain the correct set V in step (c) of the second loop of INCREMENTALCOMPILATION (Alg. 3).

Algorithm 7 Marks subgraphs when deleting a node in the IC process.

```

1: procedure REMOVENODE(Node  $X$ , MPS  $M_X$ , MPS  $M_Y$ )
2:   Delete  $X$  from  $M_X$ 
3:   Mark  $M_X$ 
4:   for all Neighbour  $M_Z \neq M_Y$  of  $M_X$  do
5:      $S \leftarrow$  separator between  $M_X$  and  $M_Z$ 
6:     if  $X \in S$  then
7:       Delete  $X$  from  $S$ 
8:       REMOVENODE( $X, M_Z, M_X$ )
9:     end if
10:  end for
11: end procedure

```

As an example, let us consider the removal of variable D from the Asia network. This operation results in the following list of modifications: (remove $E \rightarrow D$, remove $B \rightarrow D$, remove D), which is passed to Alg. INCREMENTALCOMPILATION from the edit mode. Subsequently, the moral link $(E \rightarrow D)$ is removed during the first loop of INCREMENTALCOMPILATION algorithm. Then, in the second iteration of the loop, the MPSs (DEB) and $(LBES)$ are marked. In this case, the effect of algorithm REMOVENODE is just to remove D from (DEB) . Therefore, we have to re-triangulate $G^M(\{E, B, L, S\})$, see Fig.10(c). Part (d) of the same figure shows the tree obtained from this graph. In part (e) the connecting process of the old and new structure is shown, where marked clusters are highlighted by filling them. Finally Fig. 10(f) shows the result obtained after absorbing non maximal cluster (LE) into cluster (TLE) .

4.7. Adding a node

This is a very simple operation. As X is a new variable, it will be an isolated node in the network, so, the modification consists of the addition of a new MPS/cliue

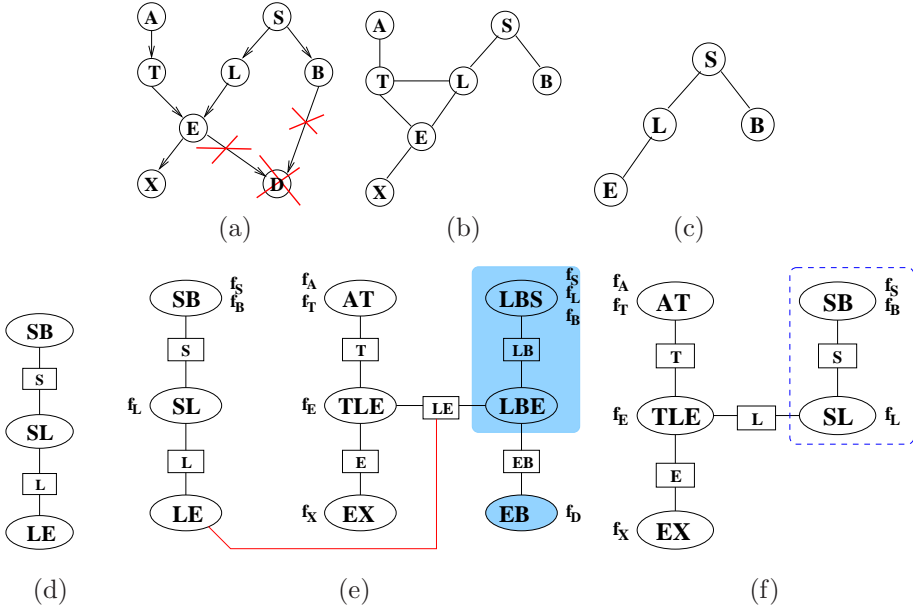


Fig. 10. MPD-based incremental compilation of Asia when removing variable D .

containing only variable X . In order to maintain single structures (trees) rather than sets (forests), we connect new clusters to the respective trees by taking any existing cluster and connecting the new cluster by an empty separator. If that this structure was used to propagate, this separator will have capacity 1 in order to store the constant number to be passed from one cluster to the other (ADDNODE, Alg. 8). The result of adding a new variable Z to the Asia network is shown in Fig. 11.

Algorithm 8 Marks subgraphs when adding a node in the IC process.

- 1: **procedure** ADDNODE(Node X)
 - 2: $C_X \leftarrow$ new created marked Clique containing only X
 - 3: $M_X \leftarrow$ new created marked MPS containing only X
 - 4: Connect C_X to \mathcal{T} by an empty separator
 - 5: Connect M_X to \mathcal{T}_{MPD} by an empty separator
 - 6: **end procedure**
-

4.8. Adding a link

Finally, we will consider the addition of a new arc $X \rightarrow Y$. This change will (at least) modify M_Y . If X is already included in M_Y , then only this MPS has to be retriangulated. Otherwise, we have to look for an MPS, M_X , in which X is included (M_X is not necessarily the MPS to which X originally was assigned). M_X and M_Y are marked and so are all the MPSs on the path between them.

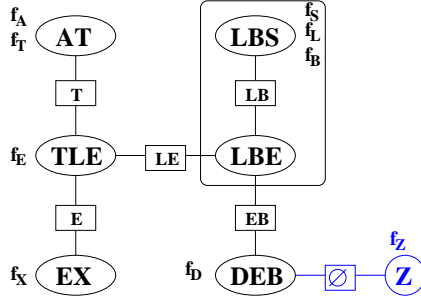


Fig. 11. MPD-based incremental compilation of Asia when adding a new variable Z .

This is the general idea of the method for adding a link. However, there is a tricky point that should be discussed. Due to the presence of empty separators, it is possible to modify the tree structure after having located M_X , in order to achieve a better (more efficient) structure for our goal. For example, if $A \rightarrow Z$ is the link to be added to the structure in Fig. 11, then we will mark all the MPSs in the tree except (EX) . However, as M_Z is connected to the tree by an empty separator, we can connect it to MPS (AT) instead. By using this new tree, only MPSs $\{(AT), (Z)\}$ have to be re-triangulated, which leads to a (far) more efficient process.

ADDLINK (Alg. 9) marks the MPSs affected by the addition of $X \rightarrow Y$.

As an example, let us consider the addition of two new links, $A \rightarrow Z$ and $Z \rightarrow X$ to the structure depicted in Fig. 11:

- (1) Adding $A \rightarrow Z$: as there is an empty separator in the path between (Z) and (AT) , the tree is modified to the one depicted in Fig. 12(a). ADDARC marks MPSs (Z) and (AT) . Also, the separator is set to A .
- (2) Adding $Z \rightarrow X$: Now there is no empty separator along the path between (Z) and (EX) , so no modification is performed over the tree. The algorithm marks (Z) , (AT) , (TLE) and (EX) as the MPSs to be re-triangulated.

Algorithm 9 Marks subgraphs when adding a (set of) links in the IC process.

```

1: procedure ADDLINK(LinkList  $L$ )
2:   for all Link  $X \rightarrow Y \in L$  do
3:      $M_X \leftarrow$  the nearest neighbour to  $M_Y$  containing  $X$ .
4:     if  $\exists$  an empty Separator  $S$  ( $S \neq \emptyset$ ) on the path between  $M_X$  and  $M_Y$  then
5:       Disconnect  $\mathcal{T}_{MPD}$  and delete  $S$ 
6:       Connect  $M_X$  to  $M_Y$  by an (artificial) Separator containing  $X$ 
7:     end if
8:     Mark  $M_X, M_Y$ 
9:     for all  $M_Z$  on the path between  $M_X, M_Y$  do Mark  $M_Z$ 
10:    end for
11:  end for
12: end procedure
    
```

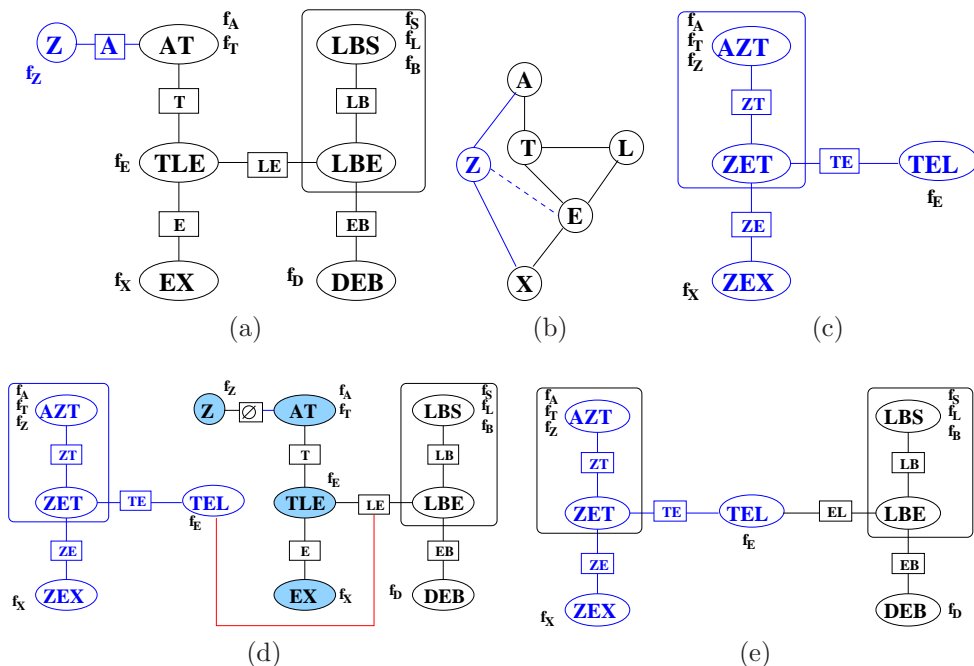


Fig. 12. Incremental compilation of the structure in Fig. 11 when adding links $\{A \rightarrow Z, Z \rightarrow X\}$.

Note that the moral link $Z - E$ has been added to G^{iM} (by MODIFYMORALGRAPH).

We get $\{(Z), (AT), (TLE), (EX)\}$ as the set of MPSs to be re-triangulated. The subgraph of G^{iM} induced by the set of variables in these MPSs ($\{Z, A, T, L, E, X\}$) yields the graph shown in part (b) of Fig. 12, and re-triangulating this we get the tree depicted in part (c) of Fig. 12. Finally, the connecting process is illustrated in Fig. 12(d) and the final result (after removing marked clusters) is shown in part (e) of the same figure.

5. Experimentation and Results

The implementation of the method described above has been integrated into the programming code of the Elvira project,^f in which several universities participate. The Elvira system is a Java tool (GUI + API) to construct probabilistic graphical models and also to evaluate new algorithms (inference, learning, etc.).¹⁰

In this section we design a series of experiments in order to study the impact of IC when modifying a BN. In most of them we restrict the compilation process to only the symbolic part, that is, clique probability tables are not reconstructed. This is enough to observe the gain provided by IC with respect to triangulation from scratch. However, we also include an experiment (Sec. 5.7) in which full compilation is carried out, in order to provide a real picture of the actual gain.

^f<http://leo.ugr.es/~elvira>

5.1. Networks and designed experiments

We have tested our approach over two different kinds of networks:

- Ten real complex networks, most of them taken from the repository[§] of the *Machine Intelligence Group* at Aalborg University, and another, *prostanet*, taken from Lacave’s research.²⁴
- A set of artificially generated networks. These networks have a slice-like structure (see Fig. 13(a)), as sometimes happens in temporal/dynamic and parametric Bayesian networks. What makes these networks interesting is the fact that every two slices (i and $i + 1$) are completely separated by the MPS $\{X_{4,i}, X_{6,i}, X_{3,(i+1)}, X_{5,(i+1)}\}$. Thus, the lower bound for the number of MPSs is $2s - 1$, s being the number of slices in the network. In order to have more complex network from the triangulation point of view (larger treewidth) we add a number of arcs inside each slice (see Fig. 13(b)). Thus, each slice can have different optimal triangulations because the intra-slice arcs are different among them and also the number of states for each variable has been randomly generated (by using a Poisson distribution of mean 4 and minimum of 2). We generated ten random networks, termed as $RbN \times S$, with N the number of variables in each slice and S the number of slices. In order to obtain dense networks, the number of intra-slice arcs have been $N - 1$ when $N \leq 10$ and $2N - 1$ when $N > 10$.

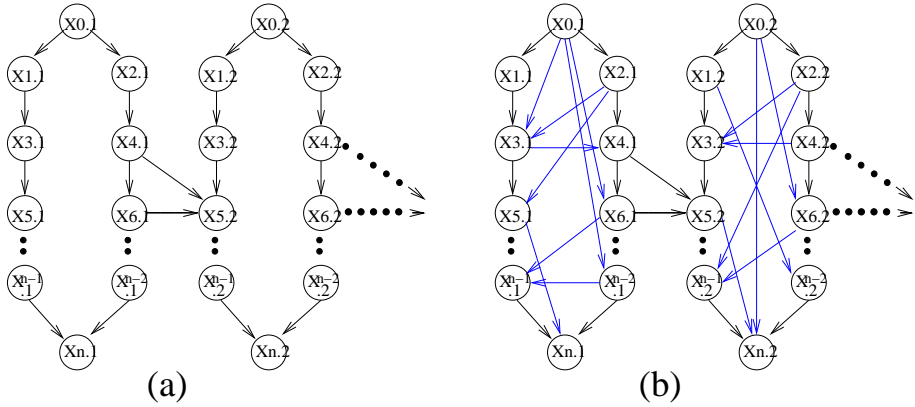


Fig. 13. Basic structure for the artificially generated networks.

To prevent the reader from being overwhelmed by too many data and graphics, we have made a selection of the most representative cases, since there is a common trend. We only show here the experiments carried out over 4 real networks and 4

[§]http://bndg.cs.aau.dk/html/bayesian_networks.html

Table 1. Description of the networks.

<i>Net</i>	$\#V$	$\mu(\#St)$	$\#E$	$\#E_M$	$\#S$	$\mu(\#V_s)$	$\sigma(\#vs)$	S_v^*
Prostanet	47	2.21	81	116	28	3.71	2.88	17
Munin1	189	5.26	282	366	70	4.14	12.61	108
Pigs	441	3.00	592	806	227	3.68	10.09	155
Munin4	1041	5.42	1397	1843	498	3.51	15.40	342
Rb10 \times 5	50	2.50	103	166	23	5.17	1.43	9
Rb10 \times 10	100	2.41	208	327	37	5.54	1.82	9
Rb20 \times 20	400	2.54	1218	2276	66	9.80	6.03	19
Rb20 \times 50	1000	2.52	3048	5743	173	9.54	5.89	19

artificial ones. We have selected this subset in order to have networks with different complexity (measured in terms of number of nodes and links). Table 1 shows some information about the networks: their name [*Net*], the number of variables [$\#V$], the mean number of states per variable [$\mu(\#St)$] and the number of links/arcs in the network [$\#E$]. Table 1 also provides information of interest for the incremental compilation process: the number of links/edges in the moral graph [$\#E_M$], the number of Maximal Prime Subgraphs [$\#S$], the mean number of variables per subgraph [$\mu(\#V_s)$] (plus the standard deviation [$\sigma(\#vs)$]), and the number of variables in the largest subgraph [S_v^*].

5.2. General idea for the experimental suite

The scene where the IC plays its main role is as follows: a user modelling a BN that has been previously compiled decides to make some changes to it. A real study with users is not feasible, so, we have made a *simulation* where we consider some of the possible situations. We carried out experiments according to the following criteria:

- (1) The four basic modifications should participate in the simulation, that is, addition/deletion of nodes/arcs.
- (2) The modifications should be *realistic*.
- (3) Both the amount (ratio) of nodes/arcs changed and their relative positions (whether they are located in the same area or not) should be studied.

5.3. Experiment 1: Random modifications

The immediate approach for generating modifications would be to randomly select nodes and arcs to add to (or remove from) a network. However, this would probably give rise to unrealistic modifications, such as linking nodes which are too far apart in the graph. Because of this, to generate a list of *realistic* modifications, we propose the procedure of choosing a set of nodes that will be firstly deleted (together with their incident nodes) creating a deletion modification list, *modListD*, and later the exactly same modifications in their inverse operation (addition) will be reversely

applied to create *modListA*. The key point here is how to choose the nodes to be modified. In this first experiment the selection of nodes has been done randomly.

Thus, we generate two modification lists which are close to being realistic, in the sense that both come from the real network. Moreover, by modifying the network in two steps, applying *modListD* followed by *modListA*, we get the original network.

Note that deleting/adding a node involves previously deleting/subsequently adding the incident links (e.g. in Fig. 2(a), deleting the node L will provoke a previous deletion of both $S \rightarrow L$ and $L \rightarrow E$). So, even when selecting only a few nodes, the impact of the modification over the network can be significant.

Finally, in this experiment 1 we get the corresponding join trees for the modified networks in two phases: (1) when deleting, modifications in *modListD* and (2) when adding, modifications in *modListA*. In both cases we perform compilation from scratch and also incremental compilation, in order to compare their results.

In this experiment, the number of variables (nChangedVars) deleted/added is controlled by the parameter n . Depending on the complexity of the network, n has a different meaning: If $\#V \leq 100$ then $n\text{ChangedVars}=n$, otherwise nChangedVars is the $n\%$ of $\#V$. Notice that even when $n = 1$, this could imply multiple modifications that will depend on how many children and parents the corresponding nodes have. In order to compare incremental compilation (IC) with traditional re-compilation, we collect different data: time (seconds), number of nodes/links affected by the modifications and size of the resulting join trees. In all the cases the same heuristic have been used for triangulation: Cano and Moral.⁴

Tables 2 and 3 show the results obtained for $n=1$ and $n=2$. The data shown in the table are: the ratio between the time required by re-compilation $[t_N]$ and IC $[t_I]$; the time required by re-compilation $[t_N]$; the number of variables $[V]$ and edges $[E]$ modified in the moral graph; the ratio^h (re-compilation/IC) of variables $[V_N^r/V_I^r]$ and edges $[E_N^r/E_I^r]$ involved in the triangulation process; and the ratio between the join tree size obtained by using re-compilation \mathcal{T}_N and IC \mathcal{T}_I distinguishing between addition and deletion modifications.

All the data are on average ($\mu(\cdot)$) over the number of runs carried out. For the number of variables involved in the triangulation process the standard deviation is also shown $[\sigma(\#V_I^r)]$. In our experiments 20 series were used, which gives rise to 40 runs, as every series produces two experiments (*modListD* and *modListA*).

From these results, with respect to CPU time, we can see that when modifications are selected randomly the gain provided by IC increases with the complexity of the BN and the MPD (number of subgraphs and number of variables per subgraph), and decreases with the modified portion of the network (parameter n). Let us postpone tree sizes details to next subsection, since conclusions are quite similar.

^hNote that this ratio is in fact the fraction of the whole network which has to be triangulated in Incremental Compilation.

Table 2. Experiment 1 (Random), $n = 1$.

<i>Network</i>	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	V	E	$\mu\left(\frac{V_I^r}{V_N^r}\right)$	$\mu\left(\frac{E_I^r}{E_N^r}\right)$	$\sigma(\#V_I^r)$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{del}$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{add}$
Prostanet	4.723	0.017	1	6.15	0.32	0.34	9.86	1.00021	1.01125
Munin1	2.654	0.095	2	9.25	0.55	0.63	23.16	1.10402	0.91866
Pigs	6.126	0.389	4	14.50	0.33	0.34	47.07	0.93279	1.05226
Munin4	1.499	1.688	10	37.41	0.37	0.43	30.50	1.10402	0.91866
Rb10 \times 5	5.477	0.017	1	8.10	0.20	0.16	2.22	1.00000	1.00000
Rb10 \times 10	7.711	0.044	2	13.65	0.19	0.15	5.20	1.00110	1.00000
Rb20 \times 20	25.714	1.260	4	57.70	0.19	0.17	8.28	1.00370	1.01389
Rb20 \times 50	35.322	9.601	10	139.50	0.18	0.16	20.68	1.00474	1.00751

Table 3. Experiment 1 (Random), $n = 2$.

<i>Network</i>	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	V	E	$\mu\left(\frac{V_I^r}{V_N^r}\right)$	$\mu\left(\frac{E_I^r}{E_N^r}\right)$	$\sigma(\#V_I^r)$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{del}$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{add}$
Prostanet	3.410	0.015	2	10.85	0.42	0.43	9.95	1.00108	1.00602
Munin1	0.998	0.087	4	17.05	0.60	0.67	3.61	1.07774	0.97465
Pigs	1.393	0.401	9	33.69	0.39	0.39	8.09	1.45111	1.48491
Munin4	1.220	1.640	21	77.60	0.42	0.47	33.16	0.99447	0.99909
Rb10 \times 5	3.060	0.016	2	15.45	0.35	0.28	4.14	1.00048	1.00000
Rb10 \times 10	4.179	0.041	4	28.40	0.37	0.30	7.72	1.00027	1.00000
Rb20 \times 20	14.062	1.273	8	110.00	0.34	0.31	15.84	1.00562	1.02527
Rb20 \times 50	19.187	9.291	20	269.55	0.32	0.30	39.16	1.00714	1.01325

5.4. Experiment 2: Modifications closer to customary usage

Although the modifications carried out in Exp. 1 seem realistic, the random selection of nodes may not reflect reality. In fact, when a user or knowledge engineer is creating or modifying a large network it is very difficult to have a broad understanding of it, or even to have the possibility of viewing the whole model. Thus, s/he usually concentrates on a limited region of the model, exploring the nodes and relations contained in that region.

To simulate this more realistic behaviour we have changed the manner in which the nodes to be modified are selected. First, we randomly select a *leaf* node X_1 from the network and all the nodes linked to it are included in a set N . This set, which will incrementally receive new elements, is the container of the next nodes which are candidates for selection. Then, at stage i , the next node X_i is randomly selected from N , and all the neighbours of X_i are added to N . In this way all the modifications are in the same region. We have labelled this experiment *neighbour* while experiment 1 is termed *random*. Tables 4 and 5 show the results of this experiment.

Table 4. Experiment 2 (Neighbour), $n = 1$.

<i>Network</i>	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	V	E	$\mu\left(\frac{V_I^r}{V_N^r}\right)$	$\mu\left(\frac{E_I^r}{E_N^r}\right)$	$\sigma(\#V_I^r)$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{del}$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{add}$
Prostanet	9.375	0.016	1	1.75	0.15	0.14	7.31	0.99408	0.99804
Munin1	1.961	0.085	2	9.15	0.57	0.65	17.35	1.08291	1.08660
Pigs	10.298	0.381	4	30.75	0.35	0.35	51.36	1.16503	1.86951
Munin4	12.590	1.703	10	31.75	0.22	0.26	149.43	1.00057	1.00164
Rb10 × 5	7.443	0.016	1	4.10	0.15	0.11	2.70	1.00000	1.00000
Rb10 × 10	12.755	0.045	2	12.20	0.10	0.07	2.73	1.00000	1.00000
Rb20 × 20	63.503	1.256	4	51.70	0.06	0.05	7.95	1.00736	1.01189
Rb20 × 50	125.137	9.787	10	102.35	0.03	0.02	14.47	0.99655	0.99768

 Table 5. Experiment 2 (Neighbour), $n = 2$.

<i>Network</i>	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	V	E	$\mu\left(\frac{V_I^r}{V_N^r}\right)$	$\mu\left(\frac{E_I^r}{E_N^r}\right)$	$\sigma(\#V_I^r)$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{del}$	$\mu\left(\frac{ \mathcal{T}_N }{ \mathcal{T}_I }\right)^{add}$
Prostanet	1.832	0.014	2	15.65	0.50	0.47	7.81	0.98297	0.99131
Munin1	1.000	0.082	4	16.35	0.59	0.67	4.08	1.02053	1.18165
Pigs	1.326	0.393	9	55.55	0.40	0.40	16.61	1.38006	0.84936
Munin4	10.616	1.654	21	67.65	0.24	0.27	158.85	1.00328	1.00036
Rb10 × 5	5.596	0.015	2	9.5	0.18	0.14	2.53	1.00000	1.00000
Rb10 × 10	9.781	0.040	4	22.25	0.12	0.09	4.46	1.00000	1.00000
Rb20 × 20	47.287	1.259	8	87.25	0.06	0.05	12.27	1.00616	1.01626
Rb20 × 50	79.440	9.288	20	163.95	0.04	0.03	17.28	1.00537	1.00087

We can then verify that with this realistic behaviour (*neighbour*), IC improves efficiency vs compilation from scratch, and now the gain obtained by IC is indeed greatly increased with respect to the *random* experiment. For example, with Munin4 and $n = 2$, even though a big fraction ($\simeq 25\%$) of the BN is affected by IC, our method is more than 10 times faster than Non-IC.

As has been pointed out, the larger the network the bigger the gain. Moreover, it is important to notice that it is precisely these large networks that require more CPU time to be compiled. As an example, a speedup of 1.326 in Pigs means that 0.296s are needed instead of 0.393s while in Munin4 the speedup of 1.499 means that less than 1.13s are needed instead of almost 1.7s. For this reason, larger networks are the ideal target for IC. However, as experiment 5 will show, the gain when considering tables construction is more relevant even for smaller networks.

When tree sizes are concerned, from experiments 1 and 2 we can see how the speed-up introduced by IC does not affect the quality of trees. In artificial networks the ratio between sizes is very close to 1.0 in all the cases. Regarding real networks, in general IC obtain a better (averaged) result, that is, ratio is greater than 1.0.

Anyway, in these cases the variation in size is more related with the use of the triangulation heuristic than with the decision of using or not incremental compilation. Thus, if we run 500 times the heuristic breaking ties randomly, we obtain the following mean and sd for the corresponding join tree sizes: Prostanet (1772 ± 0), Pigs (2614602 ± 1633493), Munin1 (243464098 ± 46393941) and Munin4 (39192069 ± 117537). Below, experiment 6 will to study this factor in greater detail and will show the evolution of the tree size when we start from a *good* triangulation.

5.5. *Experiment 3: Impact of the number/size of modifications on IC performance*

In the above experiments we modified the network by using $n = 1$ and $n = 2$, which in some cases produces a big impact on the resulting model. We also collected many statistics about the process. In this experiment we ran the same process, but instead of looking only the result after the whole process, we aim to observe the behaviour of both (re-compilation) approaches gradually. Thus, we ran the experiment for $n\text{ChangedVars} = 1, 2, \dots$ (that is, the parameter n is not used in this experiment¹). Figures 14 (real networks) and 15 (artificial networks) show the results ($\frac{t_N}{t_I}$) of this experiment averaging over 10 different runs, where $n\text{ChangedVars}$ is represented on axis X and the ratio $\frac{t_N}{t_I}$ is represented on axis Y . Note that a logarithmic scale has been used on axis Y so as to offer a finer visualisation.

From the graphics in Figs. 14 and 15 we can observe a huge speedup when only a few variables (and their links) are modified. Note that for large networks (Munin4) even modifications based on up to 30 variables yields a considerable speedup, and with 50 modifications, which are in fact too many, IC is still advantageous.

These graphics in Exp. 3 are also very useful to compare *random* and *neighbour* modifications. As we can see, in general, IC works better when realistic changes are performed.

5.6. *Experiment 4: Addition vs Deletion changes for IC*

In this experiment we wanted to compare the effect of the two types of modifications (deletion and addition) when performing incremental compilation. To do this, we based our study on experiments 1 and 2, but now we show the ratio t_N/t_I separately for runs involving deletions (type D – *modListD*) and runs involving additions (type A – *modListA*). Table 6 shows the results obtained for a representative subset of the networks used in this paper, where we distinguish between *random* and *neighbour* modes as usual. We should note that, due to the experiment design, in the deleting (*D*) phase the network is decreasing incrementally in size, because we are deleting elements (nodes together with their incident links) while in the second phase (adding – *A*) the network will grow again until adopting the original structure.

¹Notice that for example in network Munin4 $n = 1$ implies $n\text{ChangedVars} = 10$.

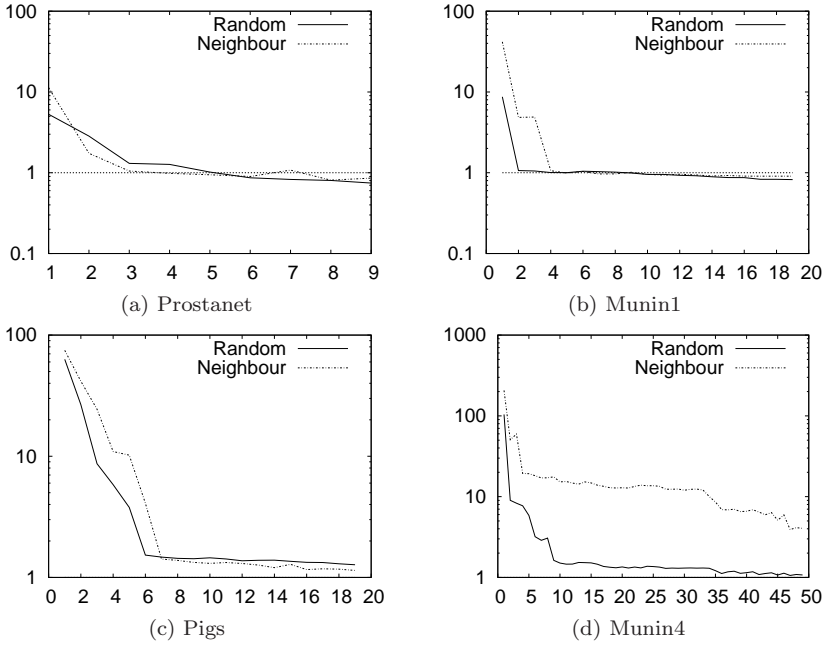


Fig. 14. Impact of nChangedVars on the ratio t_N/t_I for the *real* networks.

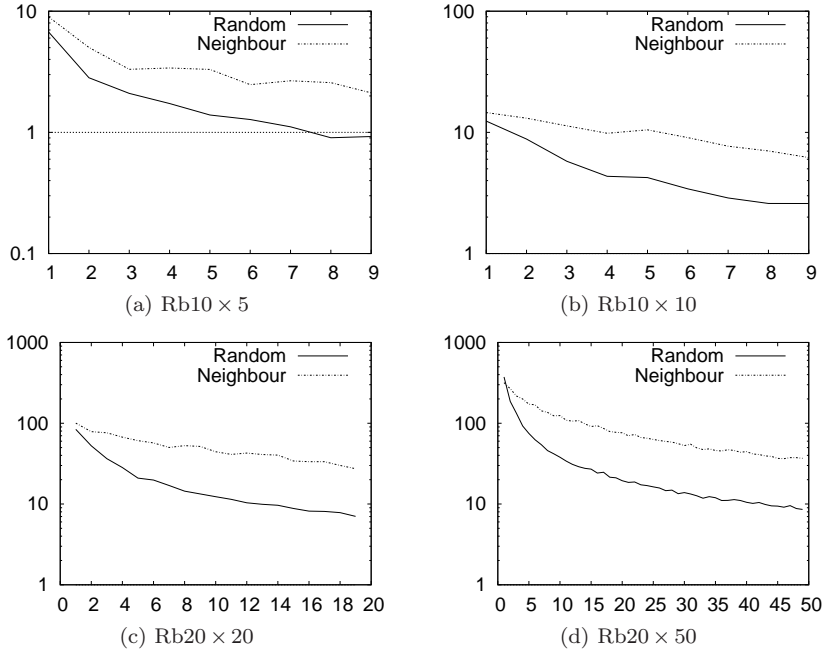


Fig. 15. Impact of nChangedVars over the ratio t_N/t_I for the *rhombus* networks.

Table 6. Results for modifications of type: deletion (D) and addition (A).

<i>Network</i> (<i>n</i>)	Random				Neighbour			
	D		A		D		A	
	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$
Munin1 (1)	2.924	0.088	2.287	0.093	3.834	0.087	1.004	0.083
(2)	0.962	0.085	0.962	0.098	0.951	0.074	1.022	0.087
Pigs (1)	9.879	0.386	1.451	0.399	9.241	0.372	7.679	0.399
(2)	1.289	0.364	1.379	0.407	1.300	0.369	1.395	0.409
Rb20 × 20 (1)	32.422	1.285	19.613	1.280	88.654	1.241	36.922	1.294
(2)	16.676	1.174	10.367	1.292	69.432	1.166	23.739	1.287
Munin4 (1)	1.566	1.705	1.459	1.719	17.364	1.693	8.307	1.744
(2)	1.377	1.692	1.237	1.742	15.395	1.714	6.184	1.750

Exp. 4 confirms that adding links (A) affects more MPSs than removing links (D), so a bigger part of the network has to be retriangulated. To check this, just compare *D* and *A* columns for both *Random* and *Neighbour* forms in all the cases. This conclusion would be stronger when the network has a homogenous MPD, as Rb20x20's results prove.

5.7. Experiment 5: Influence of IC when creating potentials

Depending on the inference method, the meaning of an initialised join tree differs. Thus, if Lazy Propagation is used,²⁹ it is enough to build the join tree and to establish the assignment of the network probability families (conditional probabilities) to the cliques in the join tree. This is the process we measured in experiments 1 to 4. However, if lazy propagation is not used (which is the case for most Bayesian network tools), then the potentials associated to each clique have to be initialised as the product of the probability families assigned to it. In this experiment we analyse the effect of using IC when potentials are initialised as probability tables, the usual representation.²⁵ Table 7 shows the data for this experiment with *n* = 1 and *n* = 2.

Table 7. Results for experiments including probability tables (clique potentials) initialisation (*n* = 1 and *n* = 2).

<i>Network</i>	Random		Neighbour		<i>Network</i>	Random		Neighbour	
	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$		$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$	$\mu\left(\frac{t_N}{t_I}\right)$	$\mu(t_N)$
Prostanet (1)	5.290	0.024	11.358	0.024	Rb20 × 20 (1)	23.068	12.194	255.876	13.040
(2)	4.064	0.026	2.319	0.024	(2)	10.985	13.567	125.006	15.206
Pigs (1)	16.687	1.219	25.931	1.250	Munin4 (1)	1.233	24.192	79.896	32.441
(2)	2.961	1.544	2.573	1.367	(2)	1.786	44.237	53.552	43.261

In this case the measures times T_N and T_I include compilation (triangulation + JT construction) together with potentials initialisation.

In this experiment we focus on real networks, although we skipped Munin1 due to the high memory resource required by the large state space of its probability tables. Of course, because of a small portion is re-triangulated in the slice-structured artificially created networks a higher benefit is obtained. We only show the results for one of these networks: Rb20x20.

In this experiment, we go beyond structure construction and we also initialise the clique potentials. This procedure involves the multiplication of probability tables which is a time consuming process for large tables such as in Munin4. That is why when recompiling with $n = 2$ Munin4 needs 0.4s (IC) vs 32s (Non-IC). IC also improves its speedup with respect to Non-IC in the Pigs network, but the improvement is smaller than in Munin4. This is caused by the fact that in Pigs all the probability tables are rather small (3 variables \times 3 states, at most).

5.8. Experiment 6: Studying IC impact on join tree size stability

In the previous experiments our main focus has been to show that using IC is faster than recompiling from scratch. We also provide data about the resulting join tree size in order to show that the reduction in time does not punish the size of the obtained join tree, that is, we do not obtain a more complex structure.

There is however a scenario that we have not tested yet. What happens if we have previously devoted some effort to produce a *good* join tree, that is, one of minimal size. In that case, it is clear that if we make some modification and recompile from scratch by using a fast triangulation method, then we loose all our previous work, but we can also expect that if IC is used, then for those parts of the graph which are not re-triangulated we maintain the *good* sub-structures in the resulting join tree and so the tree size should be smaller.

The goal of this experiment is to test if our expectation actually holds. In order to observe this behaviour gradually, we make an iterative application of compilation, that is, compiling after deleting one node, then compiling again after deleting a second one, and so until a top number (we chose 5) and then repeat the iterative process but adding nodes again one by one and in the reverse order. Modifications are generated as in previous experiments, therefore when we talk about adding/deleting a node we refer to adding/deleting also its incident edges.

The starting structure is chosen as follows: each network is triangulated 500 times by using the following heuristics: minimum fill, minimum size, minimum weight,²³ and Cano Moral.⁴ Because of random tie breaking is used we obtain a great number of different deletion sequences (and corresponding join trees). From these 2000 triangulations we choose the one leading to the join tree with smallest state space size as our starting point (σ^*).

In this experiment the results have been averaged over 10 independent runs, distinguishing between *random* and *neighbour* behaviour. In this case we focus

only on the four larger networks, being those which are more critical and variable in terms of tree size.

Figures 16 and 17 plot the output of this experiment. Points 1 to 5 in the x-axis correspond to deletion of the first, . . . , fifth variable, while points 6 to 10 corresponds to their addition in the reverse way. Thus, one should expect to see something like an inverted bell in the plots, but this fact not always happens.

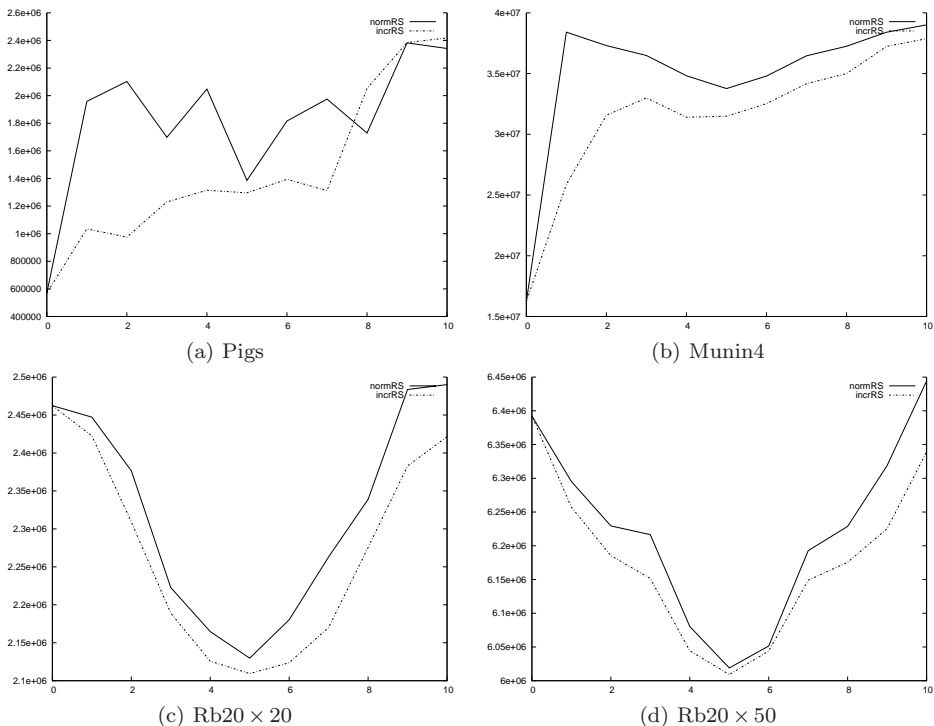


Fig. 16. Expt 6. Impact of deleting/adding variables one by one on tree size (Random).

These Figs. 16 and 17 are quite useful to understand the performance of IC from the join tree perspective, and we could analyse distinct aspects:

- (1) For real networks, with a naturally less balanced MPD,^j modifying the network will affect strongly the tree structure, which is more evident for the *Random* modifications. Note that the point 0 and 10 correspond to the same network but the resulting tree size has been increased. Because of the size of the *good* triangulations used (569835 for Pigs and 16409946 for Munin4) are too far from the average size obtained by CanoMoral heuristic, we cannot see the expected

^jAn MPD can be considered as completely balanced or uniformed when all subgraphs (MPSs) have the same number of variables.

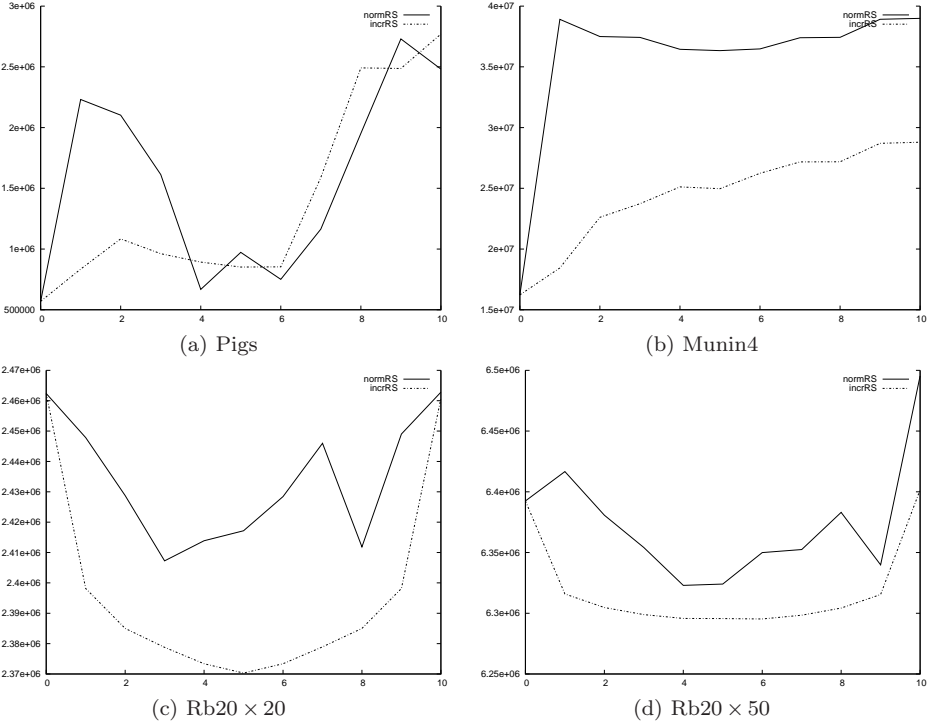


Fig. 17. Expt 6. Impact of deleting/adding variables one by one on tree size (Neighbour).

inverted bell shape, but as we can observe IC provides trees of smaller size than compilation from scratch due to the fact that we can re-use some *good* fractions from the original join tree. This behaviour is more evident when modifications are generated by using the *neighbour* approach (Fig. 17).

- (2) For the artificially generated networks, the situation is clearer. In this case, because the size of the starting *good* triangulation is closer to the average size obtained by CanoMoral heuristic, we can observe the expected inverted-bell in the plots. Thus, IC performs in a more stable way than Non-IC, even though the second is more stable than in real nets (more clear in the *Random* case) due to the simpler structure in the *RbNsS* nets. However, these results are really relevant to be studied, since they show us the ideal case for IC, when MPD decomposition is balanced and neighbour/random modifications have a clearly different behaviour.

5.9. Global conclusions from results in experiments 1 to 6

From an examination of the results obtained we are in a position to draw the following main conclusion: with respect to the parameters analysed in this study (CPU time and join tree size), Incremental Compilation is always beneficial.

When time is concerned, IC always provides better time results than compilation from scratch. Moreover, its improvement is more noticeable when the number of modifications is small, and more relevant in networks of large size. The speed-up in time is even more evident when we try to simulate a realistic experiment (*Neighbour*). Finally, if we consider the construction and itialisation of probability tables, this enhancement is even much bigger.

Tree size has also been studied in experiments 1 and 2, but especially in experiment 6. From the experiments, we can conclude that not only IC does not affect negatively in join tree structures, but in fact it favours tree stability, which was together with efficiency, one of our goals.

6. Main Conclusions and Further Work

We have developed a method for incremental compilation of Bayesian networks, as we aimed at the beginning. This method is based on MPD decomposition. So, it depends on a maximal prime subgraph representation of the graph of the Bayesian network, which is easily obtainable from the join tree representation. The key point is that the maximal prime subgraphs are the minimal subgraphs that can be triangulated independently, and the method thereby ensures that no unnecessary computations are carried out. Moreover, the method supports stability of the join tree as existing parts are recycled and only the parts that have been affected by changes in the BN are modified.

The method saves time in situations with frequently changing BNs. This is typically the case during construction and tuning of models, but also during learning of BN models minor modifications are often systematically applied. In such processes huge model spaces are searched and modifications will most often consist of addition or removal of a single arc and we have verified that for small changes IC provides a huge speedup.

In this paper, we have firstly described the technique algorithmically, and its practical impact has been backed up by an experimental evaluation of IC on a set of networks. From the analysis of the experiments carried out we can conclude that IC is an advantageous method with respect to compilation from scratch, in particular when the network is large and only slightly modified. But these two particular conditions were the ones we were really interested in solving, since large networks represent the problematic case for recompilation and the changes performed on them (when modelling or other iterative tasks) are quite usually restricted to a small proportion of the network located in a limited area.

Another reason that justifies its utility and interest is that several applications and studies that have appeared recently use and/or refer to our Incremental Compilation technique in tasks such as learning Bayesian Networks,^{16,22} computation for high-dimensional graphical models,²¹ and incremental Bayesian inference.²⁸

We reckon that this method has proved of interest even in networks having a (very) non-uniform MPD. As this is in the end a *divide and conquer* approach, its

efficiency should increase if the division is more balanced. It is for this reason that the speedup increases enormously when the network has a *nice* MPD.

As future research, we would like to continue investigating this approximate extension in order to characterise when its use is necessary. Future work also includes studying the possibility of extending this method from discrete Bayesian networks to other kinds of graphical models, such as continuous models and influence diagrams.

Finally, we would like to point out that the Incremental Compilation task can also be interpreted as a modular collaborative distribution of the network that is obtained directly from it. That is, we partition the network into subgroups that correspond to the MPSs. In the literature we find many attempts to do this modular division from the opposite point of view, that is, by constructing a total and larger network from smaller items that might be seen as subgroups or subnetworks. This is another incremental perspective, where the elements are accumulated, connected and integrated to form bigger structures. These conceptions are usually located in the framework of *Knowledge Engineering* and the *Object Oriented* philosophy. From this basis, we developed a study where the main proposals for diverse modular structures and an analysis of the connections between them and IC is done,¹¹ paying special attention to its utility for certain purposes, such as the inference process.

Acknowledgements

This work has been partially supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-67418-C03-01 and FEDER funds. The authors are indebted to the anonymous reviewers for their useful comments and suggestions.

References

1. U. A. Acar, A. T. Ihler, R. R. Mettu, and Ö. Sümer, Adaptive inference on general graphical models, in *Proc. 24th Conf. Uncertainty in Artificial Intelligence (UAI-08)*, 2008, pp. 1–8.
2. S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-tree, *Discrete Applied Mathematics* **23** (1989) 11–24.
3. O. Bangsø, M. J. Flores, and F. V. Jensen, Plug & Play Object Oriented Bayesian Networks, in *Lecture Notes in Artificial Intelligence*, **3040** (2004) 457–467.
4. A. Cano and S. Moral, Heuristic algorithms for the triangulation of graphs, in *Lecture Notes in Computer Science*, 1995, pp. 98–107.
5. A. Cano, S. Moral, and A. Salmerón, Penniless propagation in join trees, *Int. J. Intell. Syst.* **15**(11) (2000) 1027–1059.
6. A. Cano, S. Moral, and A. Salmerón, Lazy evaluation in penniless propagation over join trees, *Networks* **39**(4) (2002) 175–185.
7. J. Cheng and M. J. Druzdzel, AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks, *Journal of Artificial Intelligence Research (JAIR)* **13** (2000) 155–188.
8. A. Darwiche, Dynamic jointrees, in *UAI '98: Proc. Fourteenth Conf. Uncertainty in Artificial Intelligence*, 1998, pp. 97–104.

9. D. L. Draper, Clustering without (thinking about) triangulation, in *Proc. Eleventh Conf. Uncertainty in Artificial Intelligence (UAI-1995)*, 1995, pp. 125–133.
10. C. Elvira, Elvira: An environment for creating and using probabilistic graphical models, in *First European Workshop on Probabilistic Graphical Models (PGM-2002)*, 2002, pp. 1–11.
11. M. J. Flores, *Bayesian networks Inference: Advanced algorithms for triangulation and partial abduction*, PhD thesis, Department of Computing Systems, University of Castilla-La Mancha., 2005.
12. M. J. Flores and J. A. Gámez, Triangulation of Bayesian networks by retriangulation, *Int. J. Intell. Syst.* **18**(2) (2003) 153–164.
13. M. J. Flores and J. A. Gámez, *Advances in Probabilistic Graphical Models*, Vol. 213, *Studies in Fuzziness and Soft Computing*, chapter A review on distinct methods and approaches to perform triangulation for Bayesian networks, Springer, 2007, pp. 117–152.
14. M. J. Flores, J. A. Gámez, and K. G. Olesen, Incremental compilation of Bayesian networks, in *UAI '03, Proc. 19th Conf. Uncertainty in Artificial Intelligence*, 2003, pp. 233–240.
15. J. A. Gámez, *Advances in Bayesian Networks*, vol. 146, *Studies in Fuzziness and Soft Computing*, chapter Abductive Inference in Bayesian Networks: A review, Springer Verlag, 2004, pp. 101–120.
16. C. Gonzales and N. Jouve, Learning Bayesian networks structure using Markov networks, in *Probabilistic Graphical Models*, 2006, pp. 147–154.
17. C. Howard and M. Stumptner, Automated compilation of object-oriented probabilistic relational models, *Int. J. Approx. Reasoning* **50**(9) (2009) 1369–1398.
18. HUGIN Expert A/S, http://developer.hugin.com/documentation/API_Manuals/.API_manual_for_the_Hugin_Decision_Engine_V7.0, 2008.
19. F. V. Jensen, S. L. Lauritzen, and K. G. Olesen, Bayesian updating in causal probabilistic networks by local computation, *Computational Statistics Quarterly* **4** (1990) 269–282.
20. F. V. Jensen and T. D. Nielsen, *Bayesian Networks and Decision Graphs*, 2nd edn. (Springer Verlag, 2007).
21. B. Jones, C. Carvalho, A. Dobra, C. H., C. Carter, and M. West, Experiments in stochastic computation for high-dimensional graphical models, *Statistical Science* **20**(4) (2005) 388–400.
22. M. Karlsen and S. Pedersen, Learning Inference-friendly Bayesian Networks Using Incremental Compilation, Master's thesis, Project thesis, Group *d630a* supervised by Thomas D. Nielsen, Aalborg University, Department of Computer Science, Denmark, June 2008.
23. U. Kjaerulff, Triangulation of graphs — algorithms giving small total state space, Technical Report Technical Report R 90-09, Department of Mathematics and Computer Science. Institute of Electronic Systems. Aalborg University, March 1990.
24. C. Lacave and F. J. Díez, Knowledge acquisition in PROSTANET — a Bayesian network for diagnosing prostate cancer, in *Knowledge-Based Intelligent Information and Engineering Systems, 7th Int. Conf. (KES-2003)*, Vol. 2774, *Lecture Notes in Computer Science*, Springer, 2003, pp. 1345–1350.
25. S. L. Lauritzen and D. J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, *Journal of the Royal Statistical Society, Series B* **50**(2) (1988) 157–224.
26. S. L. Lauritzen, T. P. Speed, and K. Vijayan, Decomposable graphs and hypergraphs, *Journal of the Australian Mathematical Society. Series A*, **36** (1984) 12–29.

27. H. G. Leimer, Optimal decomposition by clique separators, *Discrete Math.* **113**(1–3) (1993) 99–123.
28. W. Li, P. Van Beek, and P. Poupart, Performing incremental Bayesian inference by dynamic model counting, in *Proc. 21st National Conf. Artificial Intelligence*, 2006, pp. 1173–1179.
29. A. L. Madsen and F. V. Jensen, Lazy propagation: A junction tree inference algorithm based on lazy evaluation, *Artificial Intelligence* **113**(1–2) (1999) 203–245.
30. S. Moral and A. Salmerón, Dynamic importance sampling in Bayesian networks based on probability trees, *Int. J. Approximate Reasoning* **38**(3) (2005) 245–261.
31. Software Corp Norsys, Netica software, <http://www.norsys.com>.
32. K. G. Olesen and A. L. Madsen, Maximal prime subgraph decomposition of Bayesian networks, *IEEE Transactions on Systems, Man, and Cybernetics. Part B*, **32** (2002) 21–31.
33. J. D. Park and A. Darwiche, Complexity results and approximation strategies for MAP explanations, *Journal of Artificial Intelligence Research (JAIR)*, **21** (2004) 101–133.
34. J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference* (Morgan Kaufmann, 1988).
35. G. Shafer and P. P. Shenoy, Probability propagation, *Annals of Mathematics and Artificial Intelligence* **2** (1990) 327–351.
36. P. P. Shenoy and G. Shafer, Axioms for probability and belief-function propagation, in *Uncertainty in Artificial Intelligence (UAI-90)*, 1990, pp. 169–198.
37. Ö. Sümer, U. A. Acar, A. T. Ihler, and R. R. Mettu, Efficient bayesian inference for dynamically changing graphs, in *Proc. 21st Annual Conf. Neural Information Processing Systems (NIPS-07)*, 2007.
38. R. E. Tarjan, Decomposition by clique separators, *Discrete Mathematics* **55** (1985) 221–232.
39. R. E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM Journal on Computing* **13**(3) (1984) 566–579.
40. W. Wen, Optimal decomposition of belief networks, in *Proc. 6th Annual Conf. Uncertainty in Artificial Intelligence (UAI-91)*, 1991, pp. 209–224.
41. Y. Xiang and J. Lee, Learning decomposable markov networks in pseudo-independent domains with local evaluation, *Machine Learning* **65**(1) (2006) 199–227.