Incremental Compilation of Bayesian networks in Practice

M. Julia Flores
Computer Science Department
University of Castilla-La Mancha
Campus universitario s/n
Albacete 02071, Spain
Email: Julia.Flores@uclm.es

José A. Gámez
Computer Science Department
University of Castilla-La Mancha
Campus universitario s/n
Albacete 02071, Spain
Email: Jose.Gamez@uclm.es

Kristian G. Olesen
Computer Science Department
Aalborg University
Fredrik Bajers Vej 7E
Aalborg 9220, Denmark
Email: kgo@cs.auc.dk

Abstract—**Inference in Bayesian networks is carried out on a secondary structure called a Join Tree. To construct this Join Tree we have to execute a process termed compilation. Compilation is a resource demanding task whose complexity increases considerably for large networks. Once the join tree has been constructed it could be possible that future modifications over the network will be necessary. In that case a whole re-compilation is needed. Changes are normally located in the same area, and if the network is large, a set of changes will typically influence only a little part of the network. For that reason we started working on a compilation that could be carried out partially. This method was first presented in [4] where basic ideas and algorithms were described. In this paper we analyse the behaviour of the Incremental Compilation in practice. To do so, we have designed a set of experiments with real and artificially generated networks.**

I. Introduction

Inference in Bayesian networks (BNs) is most efficiently carried out in a join tree, a secondary structure built of clusters of nodes in the BN. The construction of the join tree is termed *compilation*. The compilation process is important as the complexity of the inference procedures are proportional to the total size of the clusters in the join tree. For large networks the compilation procedure may be quite time consuming and many resources may be spend on compilation during construction, modification and tuning of such models, because a BN has to be recompiled each time it is modified. It is therefore beneficial to exploit methods that reuse parts of an existing join tree and only alter those parts of the join tree that correspond to modified substructures of the underlying BN. There are two main reasons for that – *efficiency* and *stability*. Efficiency is mainly obtained during modification and tuning of the BN where minor changes are typically applied to specific parts of the model, whereas major parts are left unchanged. Obviously, resources should be directed towards the construction of a partial join tree for the modified parts of the model rather than into the reconstruction of already existing partial join trees. By the reuse of existing parts of the join tree stability is also ensured and computational resources can be spent on optimising stable parts rather than reconstructing them ([1]).

In a recent paper ([4]) we devised a method for incremental compilation based on a maximal prime subgraph decomposition of the graph of a BN ([9]). The motivation (and goal) of our current work is to complete that methodological paper by investigating how the method works in practice. For that purpose IC has been integrated in the Elvira project ([2]) and this paper reports on experimental results of the method.

We will summarise the procedures for incremental compilation in the next section and in the following sections we report on the experiments. Finally, the results are analysed and we discuss the results and possibilities for further investigations.

II. MPSD-based Incremental Compilation

The proposed method for incremental compilation of BNs is based on a decomposition of the graph of a BN into its maximal prime subgraphs (MPSs). A maximal prime subgraph is a subgraph that is d-separated from its neighbouring subgraphs by complete separators. Thus, MPSs are the minimal components that can be treated independently during the compilation process. This property can be exploited for *divide and conquer* algorithms for various graph operations such as e.g. triangulations ([3]). There is a direct correspondence between MPSs and subtrees in the join tree, that is, a maximal prime subgraph always corresponds to one or more cliques in the join tree. It is this correspondence that is exploited in our method. We maintain a representation of the maximal prime subgraph decomposition (MPD) in a tree structure, $\mathcal{T}_{MPD}$, and whenever modifications are inflicted on the BN we identify and mark the MPSs that are affected by those modifications. The marked MPSs determine the parts of the join tree that have to be altered, and an updated join tree can then be constructed incrementally by replacement of only the parts of the join tree that correspond to the marked MPSs.

A. Maximal prime subgraph decomposition

The decomposition of the graph of a Bayesian networks into its maximal prime subgraphs is integrated into the well known procedure for construction of join trees for Bayesian networks. We assume the join tree construction procedure known and refer to e.g. Jensen [5] for details.

The method for identification of maximal prime subgraphs of the directed acyclic graph (DAG) G of a Bayesian network is based on a join tree representation $\mathcal{T}$ of G. First G is

moralised to obtain G^M and then G^M is triangulated to obtain G^T. A precondition for the method is that the triangulation T is minimal. This is obtained either by using a triangulation method such as the LEX M algorithm [10] that directly produces a minimal triangulation, or, alternatively, by applying the recursive thinning algorithm by Kjærulff [6] that removes redundant fill-in edges from an arbitrary triangulation. Next the cliques of G^T are identified and organised in a junction tree $\mathcal{T}$. The maximal prime subgraphs of G^M are now formed by aggregating adjacent cliques of $\mathcal{T}$ connected by a separator which is incomplete in G^M.

The following algorithm returns a join tree $\mathcal{T}_{MPD}$, where the nodes represent the maximal prime subgraphs of G^M of a Bayesian network.

Algorithm 1: ConstructMPDTree(G)

1) Moralise G to obtain G^M.
2) Find a minimal triangulation of G^M to obtain G^T.
3) Organise the cliques of G^T in a junction tree $\mathcal{T}$.
4) Aggregate all adjacent cliques in $\mathcal{T}$ with an incomplete separator in G^M to obtain $\mathcal{T}_{MPD}$.
5) Return $\mathcal{T}_{MPD}$.

Figure 1 shows the construction of $\mathcal{T}_{MPD}$ for the well-known Asia example by Lauritzen and Spiegelhalter [7]. Part (a) shows the BN for the example and in part (b) the moral graph is obtained by adding arcs between common parents of all nodes (arcs (T, L) and (E, B)), and dropping the directions of the original arcs. In part (c) the triangulated graph results from adding the arc (L, B). A resulting join tree is shown in part (d) and a check of the separators in the moral graph yields the maximal prime subgraph decomposition tree shown in part (e). Finally, part (f) gives the MPSs of the Asia network. Remark that this decomposition is unique ([9]).

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. This structural correspondence is exploited in our method for incremental compilation.

B. Modifications of a Bayesian network

There are several possible modifications that can be performed in a BN. These changes range from simple modifications, such as adjustments of numerical parameters in the (conditional) distributions for variables or adjustment of the state space for variables, to complex structural reorganisation of variables and their links, possibly altering large parts of the BN. In the present context we will concentrate on structural changes, that is, insertion and deletion of links and variables. Often such structural changes are grouped in batches and the procedure for incremental compilation is therefore designed as a two step algorithm. The first step is identifying the MPSs affected by one or more changes and the second step reconstructs the parts of the join tree affected by these changes.

C. Incremental compilation

Algorithm 2 outlines the incremental compilation procedure. The algorithm takes a list of modifications as input, thus allowing for both single modifications and batched modifications collected through the graphical interface of a tool for manipulating BNs.

Algorithm 2: IncrementalCompilation ($ModList$)

1) For each modification mod in $ModList$ do
   a) ModifyMoralGraph
   b) MarkAffectedMPSs
2) For each connected marked subtree, T_{MPS}, of $\mathcal{T}_{MPS}$ do
   a) Mark all cliques in the subtree T of $\mathcal{T}$ corresponding to T_{MPS}
   b) Construct a join tree t for T
   c) Construct a MPS tree t_{MPS} for T_{MPS}
   d) Replace T by t in $\mathcal{T}$
   e) Replace T_{MPS} by t_{MPS} in $\mathcal{T}_{MPS}$

The first loop of algorithm 2 iterates over all modifications. For each modification we adjust the moral graph. For addition and deletion of variables this has no side effects, as we assume that links connected to a deleted variable are deleted in advance and links connected to a newly introduced variable will appear in subsequent modifications in the list. Addition of new links may induce new moral links and deletion of links may remove existing moral links. Based on these modifications all affected MPSs are modified and marked. This is a non-trivial process, but we will not dig into the details here, but instead refer the interested reader to [4]. The result of this step is a MPD-tree with (possible several) connected marked subtrees. In the second step of the algorithm we iterate over these marked subtrees and adjusts $\mathcal{T}$ and $\mathcal{T}_{MPD}$. During this process the original structures of the trees may be adjusted by moving subtrees. Again, the details can be found in [4].

As an illustration of the algorithm, let us consider removal of the variable D from the Asia network. This operation is typically performed in the graphical representation of the BN and results in the following list of modifications: (remove $E \rightarrow D$, remove $B \rightarrow D$, remove D). Processing the first modification leads to deletion of the link itself and the moral link (E, B). This affects the MPSs (DEB) and $(LBES)$ (see Figure 1 (f)) and thus they are marked. Processing the second and third of the modifications simply leads to the removal of the elements in the moral graph and no further MPSs are marked. The modified moral graph corresponding to the union of all remaining nodes in the marked MPSs is shown in figure 2(a). In the second loop of algorithm 2 the cliques in the join tree that corresponds to the marked MPSs are marked. These cliques are the ones in the right hand part of figure 1(d). Next we construct a new join tree and a new MPS tree for the subgraph in figure 2(a). Part (b) of the same figure shows the join tree and the MPS tree obtained from this graph. In the current example these trees are identical as all separators are complete in the moral graph. In part (c) the result of replacing outdated subtrees with the newly created ones is shown. Observe that the clique/MPS (L, E) is not maximal, hence it is absorbed by the surviving clique/MPS (T, L, E).

III. Experimental evaluation

The implementation of the method described before has been integrated into the programming code of the Elvira

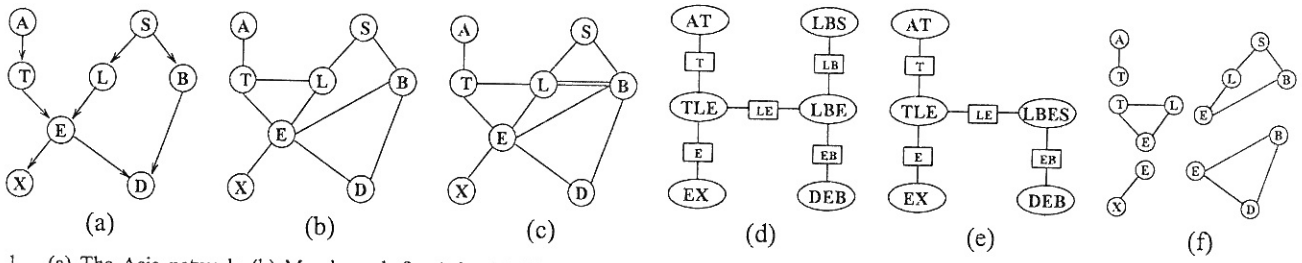

Fig. 1. (a) The Asia network. (b) Moral graph for Asia. (c) Triangulated graph for Asia. (d) A join tree for Asia. (e) Maximal Prime Subgraph Tree for Asia. (f) Maximal Prime Subgraph Decomposition for Asia.

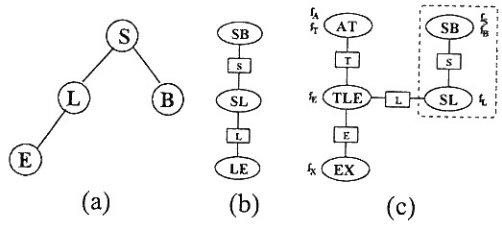

Fig. 2. MPSD-based IC of Asia when removing variable D.

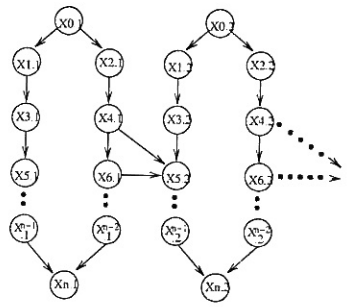

Fig. 3. Basic structure for the artificially generated networks

project[1], where several universities participate. The Elvira system [2] is a Java tool (GUI + API) to construct probabilistic graphical models and also to evaluate new algorithms (inference, learning, ...).

In this section we design a series of experiments in order to study the impact of incremental compilation when modifying a Bayesian network.

A. Networks

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

- Ten real complex networks (most of them) taken from the repository[2] of the *Decision Support Systems Unit* in Aalborg University.
- A set of artificially generated networks. These networks have a sliced-like structure (see figure 3) as some times 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 $\{X_4.i, X_6.i, X_3.(i+1), X_5.(i+1)\}$. As $\{X_1.i, \ldots, X_{n-3}.i\}$ and $\{X_{n-1}.i, X_{n-2}.i, X_n.i\}$ are the two MPSs in each slice, then the number of MPSs in a network is $2 \cdot s + (s-1)$, s being the number of slices in the network. Finally, although the structure of each slice is the same, they can have different optimal triangulations because the number of states for each variable has been randomly generated (by using a Poisson distribution of mean 4 and minimum equals to 2). We have generated ten random networks, termed as RbNxS, with N the number of variables in each slice and S the number of slices.

Because of the lack of space we only reproduce here the experiments carried out over 8 networks (4 real and 4 artificial). We have selected this subset in order to have networks with different complexity (measured in terms of number of nodes and links). Table I shows some information about the networks: its 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 [$\#E$]. Besides, table I also shows 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^*].

TABLE I
DESCRIPTION OF THE NETWORKS.

Net	$\#V$	$\mu(\#St)$	$\#E$	$\#E_M$	$\#S$	$\mu(\#V_s)$	$\sigma(\#vs)$	S_v^*
Mildew	35	17.6	46	80	15	4.8	4.25	20
Munin1	189	5.26	282	366	70	4.14	12.61	108
Pigs	441	3.0	592	806	227	3.68	10.09	155
Munin4	1041	5.42	1397	1843	498	3.51	15.40	342
Rb10x5	50	4.36	58	71	14	5.43	2.79	9
Rb10x10	100	3.82	118	146	29	5.38	2.70	9
Rb20x20	400	4.05	438	496	59	8.74	7.42	19
Rb20x50	1000	4.06	1098	1246	149	8.70	7.36	19

B. Design of experiments

The scene where the Incremental Compilation (IC) plays its main role is the following one: a user modelling a BN that has been previously compiled decides to make some changes on 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] http://leo.ugr.es/~elvira
[2] http://www.cs.auc.dk/research/DSS/misc.html

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.

Experiment 1.

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, as linking nodes which are too far in the graph. Because of this, to generate a list of *realistic* modifications, we propose the following procedure:

1. modListD = {}
2. for n=1 to numCases do
   2.1 Randomly select a node X from the network B
   2.2 modListD.append({del(X, Y) | $X \rightarrow Y$ or $Y \rightarrow X$ are in B})
   2.3 modListD.append(del(X))
3. modListA = (reverse(modListD) replacing del by add)
4. return modListD and modListA

Thus, we generate two modification lists which are 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 deleting/adding first/after the incident links (e.g. in figure 1(a) deleting the node L will provoke a previous deletion of both $S \rightarrow L$ and $L \rightarrow E$). So, even selecting a few nodes, the impact of the modification over the network can be significant.

Finally, experiment 1 consists in:

1. Let B the network, T a join tree for it, and numCases.
2. Get modListD and modListA by using the previous method.
3. B_D = modify(B,modListD)
4. Obtain T_D^I by using incremental compilation from T and T_D^N by compiling B_D from scratch
5. B_A = modify(B_D,modListA) // note that $B_A == B$
6. Obtain T_A^I by using incremental compilation from T_D^I and T_A^N by compiling B_A from scratch

In this experiment, the number of variables (numCases) deleted/added is by the parameter n. Depending on the complexity of the network, n has a different meaning: If #V $\leq$ 100 then numCases=n, otherwise numCases is the n% of #V. In order to compare incremental compilation (IC) with traditional re-compilation, we collect different data: time (seconds) and number of nodes/links affected by the modifications.

Tables II and III shows the results obtained for n=2 and n=4. 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[3] (re-compilation/IC) of variables [V_N^r/V_I^r] and edges [E_N^r/E_I^r] involved in the triangulation process. 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 also the standard deviation is shown [$\sigma(\#V_I^r)$]. In our experiments 20 series have been carried out, which gives rise to 40 runs, because every serie produces two experiments (modListD and modListA).

[3]Note that this ratio is in fact the fraction of the whole network which has to be triangulated in IC

TABLE II
EXPERIMENT 1 (RANDOM), n=2.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V_N^r}{V_I^r})$	$\mu(\frac{E_N^r}{E_I^r})$	$\sigma(\#V_I^r)$
Mildew	3.42	0.04	2	11.8	.61	.54	6.88
Munin1	**1.17**	1.24	4	17.0	.60	.67	3.58
Pigs	**1.63**	4.14	9	35.4	.40	.40	10.41
Munin4	**1.67**	25.95	21	77.6	.42	.47	32.89
Rb10x5	**3.63**	0.05	2	4.81	.40	.35	5.88
Rb10x10	**5.39**	0.20	4	9.90	.41	.35	9.82
Rb20x20	**7.96**	2.66	8	18.65	.36	.33	21.31
Rb20x50	**12.31**	21.15	20	45.89	.35	.32	54.72

TABLE III
EXPERIMENT 2 (RANDOM), n=4.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V_N^r}{V_I^r})$	$\mu(\frac{E_N^r}{E_I^r})$	$\sigma(\#V_I^r)$
Mildew	**0.71**	0.03	4	20.15	.77	.69	3.71
Munin1	**1.10**	1.16	8	34.5	.62	.69	5.86
Pigs	**1.48**	3.97	18	69.3	.43	.42	16.53
Munin4	**1.41**	24.98	42	155.3	.47	.52	36.89
Rb10x5	**1.97**	0.04	4	9.72	.62	.54	8.54
Rb10x10	**2.64**	0.18	8	20.28	.66	.57	13.32
Rb20x20	**4.49**	2.50	16	36.79	.58	.52	32.62
Rb20x50	**6.69**	19.28	40	90.58	.57	.52	73.74

Experiment 2.

Although the modifications carried out in experiment 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 wide scope 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 included in that region.

To reflect this more realistic behaviour we change 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. 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 concentrated in the same region of the network. We have denoted this experiment as *neighbour* while experiment 1 is termed *random*. Tables IV and V show the results of this experiment.

Experiment 3.

In the previous experiments we have modified the network by using $n = 2$ and $n = 4$ which in some cases produce a big impact on the resulting model. Also, many statistics about the process have been collected. In this experiment we ran the process described in experiments 1 and 2, but we only focused on the ratio $\frac{t_N}{t_I}$. On the other hand, we ran the experiment for numCases = 1, 2, ... (that is, the parameter n is not used in this experiment). Figure 4 shows the results of this experiment averaging over 10 different runs, where numCases

TABLE IV

EXPERIMENT 2 (NEIGHBOUR), $n = 2$.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V'_N}{V'_I})$	$\mu(\frac{E'_N}{E'_I})$	$\sigma(\#V'_I)$
Mildew	**0.83**	0.04	2	4.0	.65	.56	2.02
Munin1	**1.16**	1.23	4	16.5	.60	.67	5.16
Pigs	**3.98**	4.06	9	55.75	.39	.39	31.32
Munin4	**13.20**	25.93	21	82.4	.28	.32	158.38
Rb10x5	**7.15**	0.05	2	3.26	.21	.18	2.48
Rb10x10	**16.51**	0.21	4	7.19	.13	.11	5.90
Rb20x20	**55.29**	2.81	8	8.84	.04	.04	5.99
Rb20x50	**89.29**	22.70	20	25.36	.04	.03	17.09

TABLE V

EXPERIMENT 2 (NEIGHBOUR) ,$n = 4$.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V'_N}{V'_I})$	$\mu(\frac{E'_N}{E'_I})$	$\sigma(\#V'_I)$
Mildew	**0.79**	0.03	4	11.35	.69	.60	3.93
Munin1	**1.11**	1.18	8	29.25	.61	.68	9.16
Pigs	**1.36**	3.86	18	95.85	.43	.42	20.39
Munin4	**2.83**	24.37	42	178.3	.41	.47	96.63
Rb10x5	**5.04**	0.05	4	7.18	.26	.22	5.57
Rb10x10	**9.32**	0.20	8	11.75	.20	.17	8.67
Rb20x20	**28.60**	2.73	16	20.23	.08	.08	12.54
Rb20x50	**50.58**	22.07	40	48.86	.05	.05	25.65

is represented in axis X and the ratio $\frac{t_N}{t_I}$ is represented in axis Y. Note that in some cases logarithmic scale has been used in axis Y.

Experiment 4.

In this experiment we want to compare the effect of the two types of modifications (deletion and addition) when performing incremental compilation. To do this, we base 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 VI shows the results obtained for a representative subset of the networks used in this paper.

TABLE VI

RESULTS FOR MODIFICATIONS OF TYPE: DELETION (D) AND ADDITION (A).

	Random				Neighbour			
	D		A		D		A	
Network (n)	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$
Munin1 (2)	1.18	1.18	1.16	1.31	1.15	1.16	1.16	1.30
(4)	1.08	1.01	1.11	1.31	1.10	1.05	1.12	1.31
Rb20x20 (2)	9.55	2.47	6.36	2.85	74.44	2.77	36.12	2.85
(4)	5.41	2.11	3.57	2.89	40.14	2.61	17.06	2.86
Munin4 (2)	1.72	25.07	1.63	26.84	16.21	25.02	10.19	26.84
(4)	1.37	22.77	1.44	27.19	3.06	21.78	2.60	26.97

Experiment 5.

Compilation of a BN is the process that takes a network as input and produces an initialised join tree, that is, a join tree over which inference can be carried out. Depending on the inference method, the meaning of an initialised join tree differs. Thus, if Lazy Propagation is used [8] 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 just the process we have 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

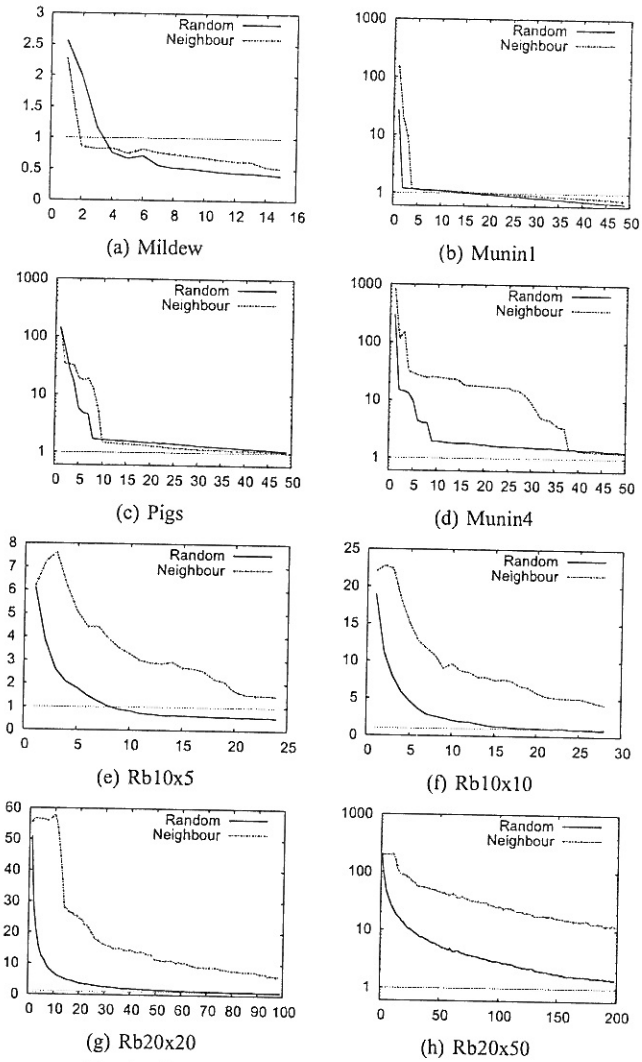

Fig. 4. Impact of numCases over the ratio t_N/t_I

to it. In this experiment we analyse the effect of using IC when potentials are initialised as probability tables (the usual representation, e.g. [7]). Table VII shows the data for this experiment with $n = 2$. Artificial networks have not been considered since their structure provokes that nearly all the cliques contain only one family. So, there is no multiplication of tables and we would be in the same case as exp. 1 and 2. For real networks, we skipped Munin1 due to the memory resource required by the large state space of its probability tables.

TABLE VII

RESULTS FOR EXPERIMENTS INITIALISING TABLES ($n = 2$).

	Random		Neighbour	
Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$
Mildew	84.33	18.17	1.03	18.17
Pigs	2.46	7.35	5.03	6.49
Munin4	1.22	157.93	22.75	164.64

IV. ANALYSIS AND DISCUSSION

In addition to the data shown in previous tables, we have analysed if there are statistically significance difference,

with respect to CPU time, between Incremental and Non-Incremental Compilation by using a paired t-test (significance level α=0.05). When statistical difference is found, values in column $\mu(\frac{t_N}{t_I})$ (tables II, III, IV and V) are written in boldface. Note that there is only one case where statistical difference is not observed (Mildew, *random*, $n = 2$).

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

A careful examination of the experimental results leads us to a more specific analysis:

- When modifications are selected randomly (exp. 1) 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 the modified portion of the network (parameter n).
- When a real behaviour is simulated (*neighbour*) the previous observation also holds and now the gain obtained by IC is severely increased. For example, with Munin4 and $n = 2$, even though a big fraction ($> 25\%$) of the BN is affected by IC, our method is 13 times faster than Non-IC.
- As has been pointed out the larger the network is the bigger the gain is. Moreover, it is important to notice that precisely these large networks are those that require more CPU time to be compiled. As an example, a speedup of 16.5 in Rb10x10 means that 0.01s are needed instead of 0.2s while in Munin4 the speedup of 13.2 means that less than 2s are needed instead of 26s. For that reason, larger networks are the ideal target for IC.
- In experiment 1 and 2 a moderate number of modifications has been considered. The purpose of experiment 3 is to illustrate how IC behaves as a function of the number of modifications. From the graphics shown in figure 4 we can observe a huge speedup obtained when only a few variables (and their links) are modified. Remark that for large networks (Munin4) even modifications based on up to 30 variables yields a considerable speedup.
- The 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.
- Exp. 4 has confirmed that adding links (A) affects more MPSs than removing links (D), so a bigger part of the network has to be retriangulated. This conclusion is stronger when the network has a homogenous MPS Decomposition, e.g. Rb20x20.
- In exp. 5 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 Munin4. That is why recompiling Munin4 needs 7s (IC) vs 3 minutes (Non-IC). IC also improves its speedup with respect to Non-IC in Pigs network, however 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).
- The results obtained for Mildew do not fit for many of the previous remarks. In fact, Mildew is a quite particular BN since it only presents one leaf node. This node has only one parent that is indeed the biggest one (100 states!), which is included in the largest MPS (20 of the total 35 variables). Thus in the *neighbour* case we are always (except for one single node) changing the (by far) biggest subgraph whereas *random* experiments could avoid it. As expected, this circumstance is considerably emphasised if we include the table initialisation.

V. Conclusions and Future Work

In this paper an experimental evaluation of Incremental Compilation of Bayesian networks has been presented, completing the methodological work described in [4]. 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 slightly modified. This method has been proved of interest even in networks having a (by far) non-uniform MPS Decomposition. Of course, the speedup increases enormously when the network has a nice MPSD.

For further research, we plan to test the behaviour of IC vs Non-IC in terms of stability, i.e. how (dis)similar the structure of the obtained Join Tree is compared to the initial one.

Acknowledgment

This work has been partially supported by the Spanish MCyT, under project TIC2001-2973-C05-05.

References

[1] D.L. Draper. *Clustering Without (Thinking About) Triangulation.* Proc. of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, 125–132. Morgan Kaufmann, 1995.

[2] Elvira Consortium. *Elvira: An Environment for creating and using probabilistic graphical models.* Proc. of the First European Workshop on Probabilistic Graphical Models (PGM'02) , Cuenca, Spain. *1–11*, 2002.

[3] M.J. Flores and J. A. Gámez, *Triangulation of Bayesian networks by retriangulation.* Int. J. of Intelligent Systems, **18**:2, *153-164*, 2003.

[4] M. J. Flores, J. A. Gámez and K. G. Olesen. *Incremental Compilation of Bayesian networks.* Proc. of the 19th Annual Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann. *233–240*, 2003.

[5] F.V. Jensen. *Bayesian Networks and Decision Graphs.* Springer-Verlag, New York, 2001.

[6] U. Kjærulff, U. *Aspects of efficiency improvement in Bayesian networks.* Ph.D. thesis, Dep. of Computer Science, Aalborg Univ., Denmark, 1993.

[7] 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, B., **50**, *157-224*, 1988.

[8] A.L. Madsen, and F.V. Jensen. *LAZY propagation: A junction tree inference algorithm based on lazy evaluation.* Artificial Intelligence **113**, *203–245*, 1999.

[9] K. G. Olesen and A. L. Madsen. *Maximal prime subgraph decomposition of Bayesian networks.* IEEE Transactions on Systems, Man and Cybernetics, Part B. **32:1**, *21–31*, 2002.

[10] D. J. Rose, R. E. Tarjan and G. S. Lueker. *Algorithmic aspects of vertex elimination on graphs.* SIAM Journal of Computing, **5**, *266-283*, 1976.