UNED

Escuela Técnica Superior de Ingeniería Informática

Máster en Inteligencia Artificial Avanzada:
Fundamentos, Métodos y Aplicaciones

An empirical comparison of Influence Diagrams algorithms

Miguel Ángel Artaso Landa

Advisors:
Prof. Manuel Luque Gallego
Prof. Francisco Javier Díez Vegas

Madrid - September 2014

Acknowledgement

I want to express my gratitude to every person that has helped me to make this dissertation to come into being. First of all, I would like to give a warmth thanks both to Manuel Luque and Francisco Javier Díez for their incalculable contributions. And also to Manuel Arias and Iñigo Bermejo for their help to get out of some jams. Finally, to my family, for their constant support through all my life.

Abstract

Probabilistic Graphical Models (PGMs) are widely used in many domains when reasoning with uncertainty. They are used to obtain the maximum expected utility and the optimal policy—the best decisions—in different scenarios. When dealing with real-world problems, the model built can be quite complex. As not all the algorithms perform the inference with the same efficiency, it is important to know which one is better to apply depending on the circumstances. Therefore it is important to compare the performance of the those algorithms for different models.

In this Master Thesis we compare four inference algorithms for influence diagrams (IDs): variable elimination, arc reversal, strong junction tree, and the conversion into a LIMID. For our experiments we have used OpenMarkov[1], an open software tool developed by the Research Centre for Intelligent Decision-Support Systems (CISIAD) of the UNED. The first algorithm was already implemented in this tool; the other three have been implemented by the author of this thesis. We have also programmed the generation of different IDs that have been used to compare the algorithms. We then have confronted the computational time and the memory used by the algorithms when facing these IDs and analysing the results of the experiments we give some recommendations about which algorithm to use depending on the structure of the model.

List of keywords

- Probabilistic Graphical Models
- Bayesian Networks
- Influence Diagrams
- LIMID
- Inference algorithms
- Empirical comparison

[1] `http://www.openmarkov.org/`

Resumen

Los Modelos Gráficos Probabilistas (MGP) son ampliamente usados en diferentes dominios donde hay que tratar con incertidumbre. Se emplean para obtener la máxima utilidad esperada y la política óptima, las mejores decisiones, en diferentes escenarios. Cuando se afrontran problemas de la vida real, el modelo final que los representa puede ser harto complicado. Debido a que no todos los algoritmos de inferencia son igual de eficientes, es importante saber cuál es mejor aplicar dependiendo de las circunstancias. Por lo tanto, es importante comparar la eficiencia de los algoritmos cuando ante diferentes modelos.

En este Trabajo fin de máster, comparamos cuatro algoritmos para diagramas de influencia: eliminación de variables, inversión de arcos, árbol de uniones fuerte y conversión a LIMID. Para nuestros experimentos hemos utilizado OpenMarkov[2], una herramienta de código abierto desarrollada por el Centro de Investigación sobre Sistemas Inteligentes de Ayuda a la Decisión (CISIAD) de la UNED. El primer algoritmo estaba ya implementado en esta herramienta; los demás han sido implementados por el autor de este trabajo. Además, hemos implementado la generación de diferentes diagramas de influencia, que después han sido utilizados para comparar los citados algoritmos. Después, hemos contrastado el tiempo y la memoria empleados por los algoritmos en la inferencia de estas redes y el análisis de los resultados nos ha llevado a dar algunas recomendaciones sobre qué algoritmo utilizar dependiendo de la estructura del modelo.

Lista de palabras clave

- Métodos gráficos probabilistas
- Redes bayesianas
- Diagramas de influencia
- LIMID
- Algoritmos de inferencia
- Comparación empírica

[2] `http://www.openmarkov.org/`

Contents

List of figures

List of tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Almost any knowledge has uncertainty so there exists uncertainty in almost any real life problem (Tversky & Kahneman, 1974). Image, natural language and document processing, economics, social sciences, education and bioinformatics. All these real world domains, and they are far from being the only ones, are rich in uncertainty. A knowledge representation system cannot be just boolean. Things are not completely true or false. To reason with uncertainty, we need probabilistic models to represent how likely something is true or false.

Probabilistic models were developed to shape the uncertainty. However, modelling with them many real-world problems, even some small ones, becomes difficult. Probabilistic graphical models (PGM) offer a way to represent, infer and learn from probabilistic models. The basic structure of a PGM is a graph in which conditional (in)dependencies are shown through arcs or the lack of these between nodes. Along with the graph, the PGM will include a Conditional Probability Distribution (CPD) of the probabilistic model that is representing.

Within the PGM framework, various models have been developed, each one of them intended to be used in different areas of study, according to their purposes. Therefore, a PGM can have different structural characteristics, according to the use it will have. In a PGM graph there may be directed and undirected arcs, and different types of nodes: chance, decision and utility. Those with only directed arcs connecting their nodes are called directed graphical models. These are suitable to represent causal relationships between their variables. The main types of directed PGMs are Bayesian networks (BNs) and influence diagrams (IDs), where the former is a particular case of the latter. Both

of them have a directed acyclic graph (DAG) whose nodes represent chance variables. In IDs we can find also decision and utility variables. BNs are extensively used for diagnosis problems whereas IDs are used as decision analysis tools (Barber, 2012).

When building a model, we could have information about the structure (graph) and the system could be fully or partially observable. Learning about the structure and/or the CPD, the probability distribution representing the model, can emerge from data. Although sometimes approximate, once a problem is modelled using the representation indicated above, the PGM can be used to make some inferences; through the use of the inference and evaluation algorithms it is possible to estimate the values of those nodes of which our knowledge is limited. Several tools have been developed—such as OpenMarkov, Genie, Hugin, Elvira, to name a few—that can model a problem and evaluate it through different algorithms.

When the inference algorithms have to deal with a real-world problem, not all of them perform equally well. To evaluate an inference algorithm performance, there exist two computational complexity indicators: time and space (Knuth, 1997). The time complexity is how much does it take to the algorithm to complete the inference task. The space complexity means the highest peak in the amount of memory used by the algorithm during the inference process.

Regardless of the remarkable number of inference algorithms that have been developed, it is difficult to find comparisons of the performance of the most important inference algorithms about their efficiency regarding time and space complexities. What is more, when they have been addressed it has been typically confronting only two algorithms and with few examples. The comparison of these complexities in different scenarios, would help to decide which algorithm is best to use according to the particular circumstances of the problem in evaluation. Even more since the inference of PGMs is a NP-complete problem (Cooper, 1988). This thesis will be focused on comparing and contrasting the performance of different algorithms solely using problems modelled with IDs directed graphical models.

1.2 Objectives

As stated in Section 1.1, the main objective in this thesis is to compare and contrast the performance—time and space complexity—of different algorithms facing problems that have been modelled using IDs. This comparison will be made in the interest of answering the following questions: Why do algorithms perform different when facing the same ID?

Is it a matter of the structure? Is it because of the data? The tools that are developed to model and evaluate PGMs and are capable of using IDs can be benefited from the answers to these questions; according to the information inferred from the comparison, these tools will be able to distinguish which algorithm performs better on different real-world models. Moreover, the conclusions obtained can be applied to other PGMs than IDs, such as dynamic models. These models include DLIMIDs and POMDPs among others. For their ability to model problems that evolve in time, these PGMs are more powerful than IDs. Thus, the comparison that is going to be made can be utilised in a range of tools and models to choose which algorithm fits better in a certain real-world problem.

Only IDs with no super value nodes—value nodes with no children—will be accepted in the algorithms implemented. If an ID with more than one value node is presented to the algorithm, the expected utility will be the sum of the single expected utilities. And regarding the algorithms, the following of exact inference type are the ones that are going to be evaluated:

1. Variable elimination
2. Arc Reversal
3. Strong junction trees
4. Conversion to LIMIDs

Three of these algorithms, all but Variable elimination, need to be implemented in OpenMarkov. This will be another objective in this thesis. The Artificial Intelligence community will be benefited since almost none of them are available as open source.

1.3 Methodology

The investigation was conducted under the framework of the open source application OpenMarkov. This tool is developed in the CISIAD at the UNED. It is implemented in Java 1.7, and stores the probabilistic networks in the ProbModelXML format (Arias et al., 2012)—see Section 3.2.1 for further details on it. As we expressed in Section 1.2, we coded three out of the four algorithms that we compared. To compare them, we used two different kinds of networks, which we explain in Chapter 3.

We have also developed this dissertation with the aim of providing a way in which the research done on it is reproducible. Making research reproducible and making the

development in an open source tool like OpenMarkov, allows researchers not only to reproduce and check the data but also to contribute to the research—see 3.7 for details.

1.4 Organisation of the Master Thesis

The layout of this master thesis is as follows. First, we will present the state of the art regarding the algorithms that we have used for the comparison. Then, we will discuss the development of the algorithms, followed by their comparison which is presented along with the applications of the research. We will end this document describing the future work and summarising the results.

Chapter 2

State of the art

In this chapter we present the basic concepts and the state of the art of influence diagrams—a type of Probabilistic Graphical Models—and four algorithms that evaluate them. Their implementation and behaviour facing the inference of different types of probabilistic networks, will be the core part in the remainder of this dissertation presented in the subsequent chapters.

2.1 Probabilistic Graphical Models

The Probabilistic Graphical Models (PGMs) framework has been developed based on declarative representation (Koller & Friedman, 2009), that splits the knowledge from the reasoning. Using this approach, the framework is capable to host and model problems of different natures and domains. Since real-world applications lack of full certainty, this framework incorporates the probability theory as joint probability distributions associated with some nodes of the model. Combining different kinds of techniques and algorithms, PGMs allow manipulating high-dimensional scenarios that classical tools cannot cope with.

2.2 Bayesian networks

A Bayesian network (Pearl, 1988) (BN), is a probabilistic graphical model that can have only one kind of node, chance nodes. We will refer to these types of nodes also as random nodes, indistinctly. A real-world scenario problem may contain only events that the decision maker can not control; it is modelled just with chance nodes. A Bayesian network has three elements: a set of chance nodes, an Acyclic Directed Graph (ADG)

$G = (X, A)$, and a probability distribution over the set of nodes that can be factored as:

$$P(x) = \prod_i P(x_i|pa(X_i)) \tag{2.1}$$

where $P(X_i|pa(X_i))$ is the joint probability derived from $P(X)$. Given that each node represents a variable, we will use indifferently the terms variable and node. The graph is linked to the probability distribution which means that the graph determines how the probability is factored. A link between two nodes, $A \rightarrow B$, means that the variables may be correlated probabilistically.

When we are watching a system and the value of a chance node is known, the variable is said to be observed and is part of the evidence of the model. The probability of a chance node when evidence has yet to be taken into account is called *prior* potential, and *posterior* potential otherwise (Kjærulff & Madsen, 2010).

2.3 Influence diagrams

While Bayesian Networks contain just one kind of node, chance nodes, an influence diagram (Howard & Matheson, 1984) (ID), may contain two more kinds of nodes: decision nodes and utility nodes—the latter can be ordinary or super utility nodes. Subsequently, the BNs are a particular case of the IDs, where only one type of variable is present. The decision nodes allow representing actions under the direct control of the decision maker. When the IDs were developed the utility nodes had no children. Eventually, this limitation was relaxed and the utility nodes were allowed to have children, only of utility type. The former where then called ordinary and the latter super value nodes (Tatman & Shachter, 1990). A super value node combines the values of its parents, whether aggregating or multiplying them.

In the influence diagrams, due to the increase in the types of nodes with respect to Bayesian Networks, the links have different meaning, depending on which nodes they connect. A link that emerges from a decision node D_i and that ends in another decision D_j implies that D_i precedes D_j in time. Whereas if the link that finish in D_j has begun in a chance node C, the link tells us that the variable C is known when the decision has to be made. Finally, arcs into utility nodes represent functional dependency. If as described in the previous subsection, random nodes have a probability distribution associated, utility nodes have a real-valued function that maps each configuration of parents onto a real number; for super value nodes, it is a utility-combination function. We will refer also to this probability distribution or real-valued functions as potentials.

A policy in a decision node (δ_d) specifies the values of the decision node for each configuration of its parents. The union of all the policies in an ID conforms the strategy (Δ) of the model.

There exist two assumptions that are generally associated with IDs, and that we will take in this dissertation: the decision ordering and the no-forgetting assumptions. The first one establish that there is a total ordering of the decisions, showing the order in which decisions are made. This assumption leads to the second one; if a chance variable C is linked to a decision variable D_j and thus C is known when the decision maker chooses over D_j, then C is also known to any decision node D_k that succeeds D_j.

2.4 Inference in influence diagrams

The aim of applying an inference algorithm on an ID is to seek for an optimal configuration of policies that maximises the expected utility—an optimal strategy. We refer to the set of chance nodes as $\mathbf{V}_C$ and to the set of decision nodes as $\mathbf{V}_D$. Then, the formal definition for the strategy is

$$P_{\Delta}(\mathbf{V}_{C,}, \mathbf{V}_D) = \prod_{C \epsilon \mathbf{V}_c} P(c|pa(C)) \prod_{D \epsilon \mathbf{V}_D} \delta_d(d|pa(D)) \tag{2.2}$$

With this formula, and knowing that ψ is the potential of the utility nodes, the maximum expected utility of a strategy is defined as follows

$$UE(\Delta) = \sum_{\mathbf{V}_c} \sum_{\mathbf{V}_D} P_{\Delta}(\mathbf{V}_{C,}, \mathbf{V}_D)\psi(\mathbf{V}_{C,}, \mathbf{V}_D) \tag{2.3}$$

With all the above, the optimal strategy, the one that maximises the expected utility the can be expressed as:

$$\Delta_{optimal} = argmax_{\Delta \epsilon \Delta *} UE(\Delta)$$

In the path to make it out, it may happen that not all the potentials or links are needed in the inference process. Depending on the structure of the probabilistic network, some nodes may be removed without losing any relevant information. Barren nodes (Shachter, 1988) have no descendants and their potentials have no variables of our interest. These nodes can be eliminated from the graph.

On the other hand, some decisions may have non requisite links headed to them. In a Probabilistic Graphical Model, a path between two (set of) nodes A and B can be blocked

by a set of observations S, and if so the nodes are *d-separated* (Pearl, 1988).

$$A \perp B|S$$

If certain conditions of *d-separation* are met (see algorithm 2.4.4 for details) a node is not a requisite for a certain decision, and is safe to remove their link. Processing the model looking for this kind of nodes and links can be cost-efficient, but not all the algorithms take advantage of them.

Summing up, to calculate an optimal strategy and consequently the maximum expected utility of an ID, we can carry out two main operations over the potentials, apart from the multiplication. The first one is to determine the marginal probability of a chance node to remove it from the graph. We will call it in the algorithms, marginalisation.

$$P(X') = \sum_i P(X)$$

The other operation, maximisation, has been described above and is used when a decision is alienated from the graph

$$argmax_D \Delta$$

2.4.1 Variable elimination algorithm

This is a simple algorithm, one of the earliest to be applied to the inference of PGMs and it is hard to say who implemented it. First, it is established a strong elimination order (Jensen & Nielsen, 2007). Since the operations described at the beginning of this section (marginalisation and maximisation) do not commute, they have to be applied in a certain order that comes from the structure of the model. The temporal order of the variables imposes limitations. In the temporal order, the decisions are like milestones that split the chance nodes that lay between them in the graph. In the elimination order, only the decisions have a fixed position, whereas to the chance nodes between two decision nodes a heuristic is applied to choose their order. There are several heuristics for this that try to reduce the number of operations to be made, as the complexity grows exponentially. In the variable elimination algorithm, the time and space costs depend on the size of the largest operand created during the inference.

The potentials remaining have the desired information. In the Figure 2.1 it can be found an example of the variable elimination algorithm applied to an influence diagram.

Algorithm 2.1 Variable elimination algorithm.

```
1: eliminate barren nodes in the influence diagram
2: S = variable elimination order calculation
3: L = list of potentials to be returned
4: while !S is empty do
5:    s = the first variable to be eliminated from S
6:    Remove s from S.
7:    C = chance network potentials related to s
8:    U = utility network potentials related to s
9:    Remove C and U from the network
10:   if s is a chance node then
11:      C = marginalize and multiply the potentials C over s
12:      U = marginalize and multiply the potentials U over s
13:   if s is a decision node then
14:      C = maximize and multiply the potentials C over s
15:      U = maximize and multiply the potentials U over s
16:   The lists C and U are added to L
17: L is returned
```

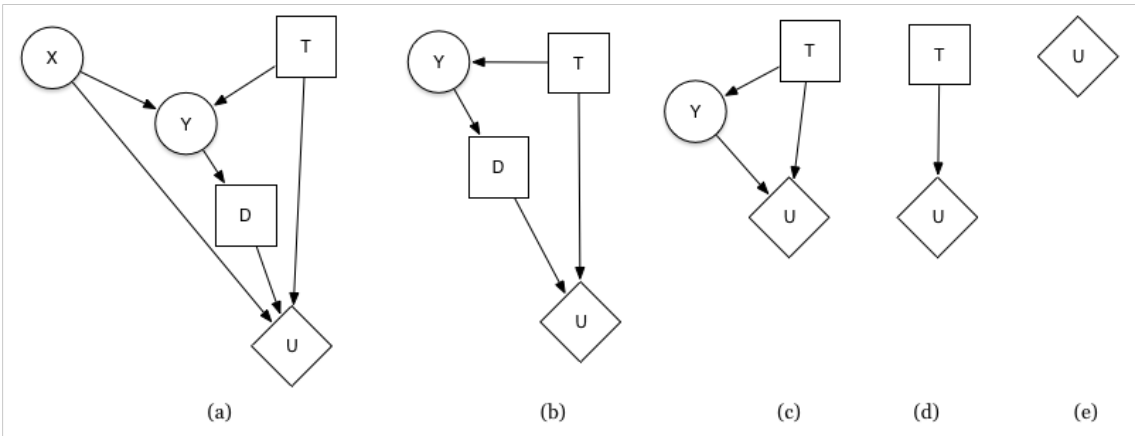

Figure 2.1: An example of the Variable Elimination inference. The figure (a) is an influence diagram and from (b) to (e) we can see in each step one variable eliminated, in the order X, D, Y and T.

2.4.2 Arc Reversal algorithm

The first implementation of the Arc Reversal algorithm was proposed by Shachter (1986). This algorithm evaluates the influence diagram directly, but only copes with models that have just one utility node. Tatman & Shachter (1990) proposed an algorithm that deals with networks that host super value nodes, but only when all the values are positive. The basic idea that underlines in the algorithm is to eliminate nodes, sometimes modifying the arcs in the network and their related potentials. Nodes are consecutively eliminated

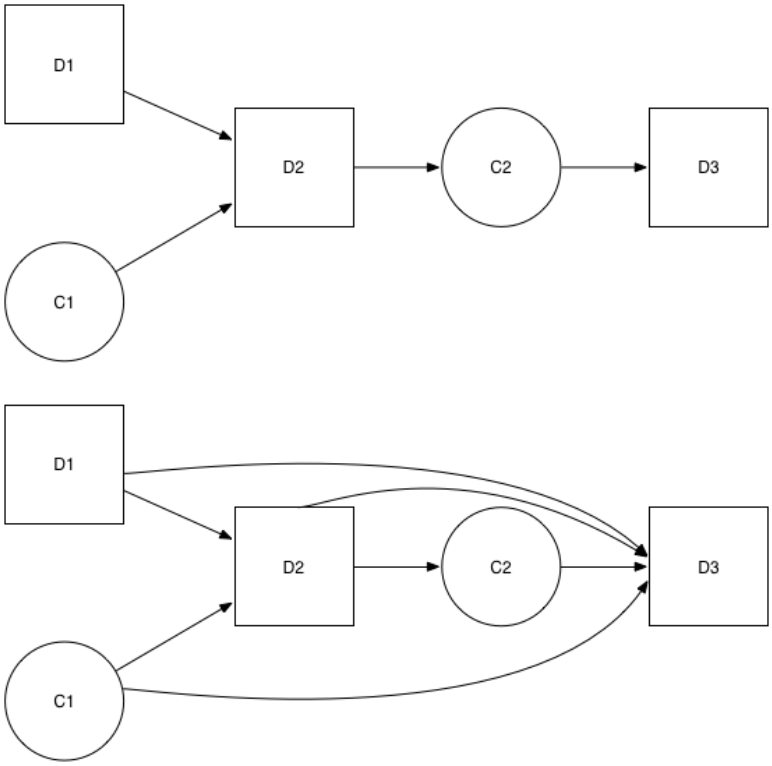

Figure 2.2: Adding "no forgetting" arcs.

from the network and eventually there will be no remaining nodes. At this point, both the strategy and the maximum expected utility are calculated.

Before the model is evaluated, some transformations must be done on it. On the one hand, when a decision node precedes temporally another one, the informational predecessors (parents) of the former have to be also informational predecessors of the latter. This process is shown in Figure 2.2. On the other hand, all the barren nodes are removed from the model. Besides the operations described before—maximisation and marginalisation—, there are three additional operations that may be applied to the ID to calculate the strategy and the maximum expected utility. Two of them pursue the elimination of one kind of node in the graph and below we show to the organisation of the model after the deletion of such nodes. We explain also below the circumstances in which a certain node is eligible to be deleted. The Figures 2.3 and 2.4 picture which links remain and/or which links are created when removing a chance node and a decision node, respectively. In the figures, the cloud represents a group of nodes, where CP states for conditional predecessors—parents—of chance or decision nodes and IP states for informational predecessor—parents—of utility nodes.

The algorithm is named after the third operation that is applied on the model, arc

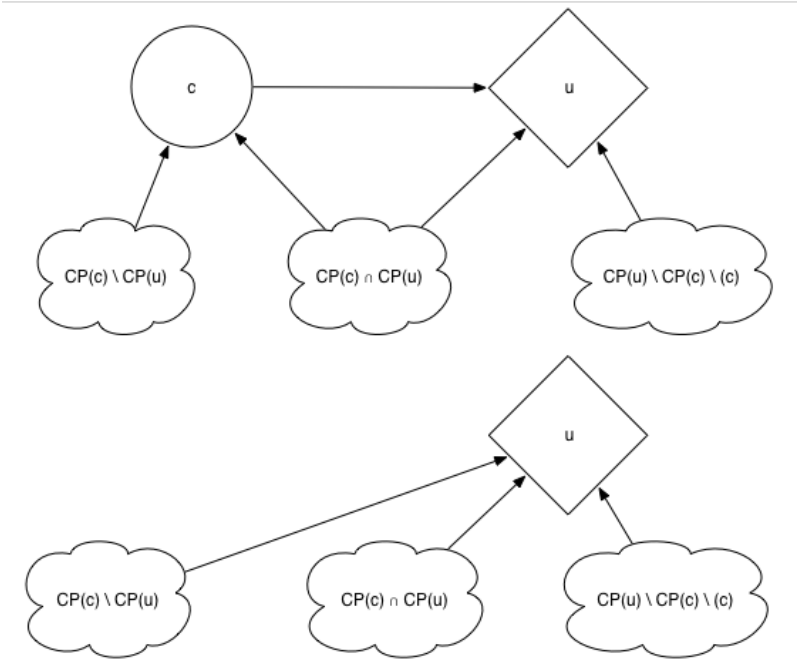

Figure 2.3: Example of removing a chance node.

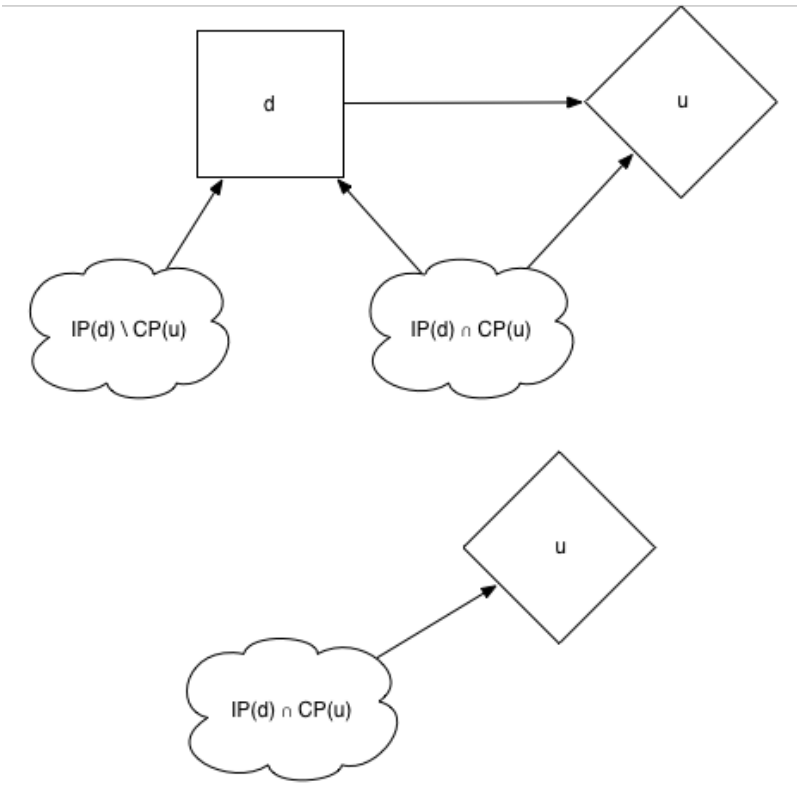

Figure 2.4: Example of removing a decision node.

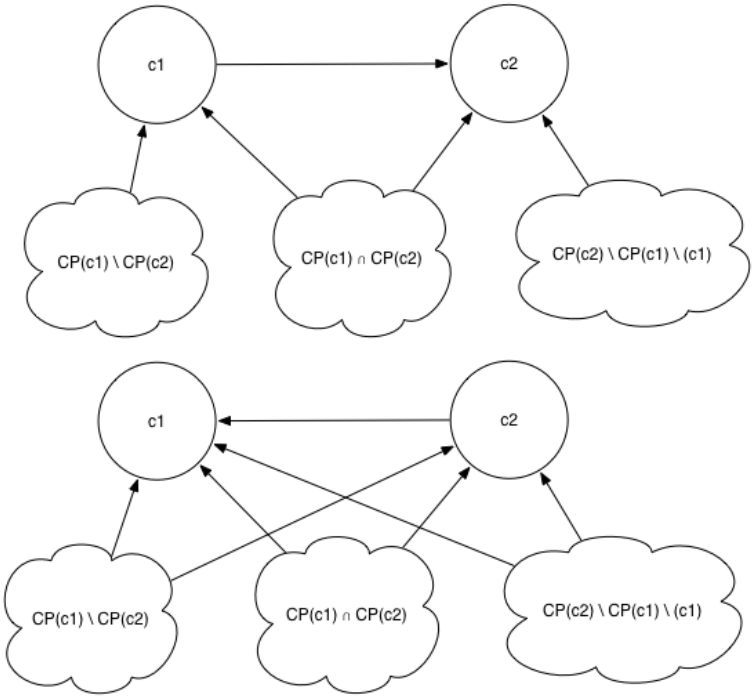

Figure 2.5: Example of reversing an arc.

reversal. It changes the orientation of a link and in order not to lose any information, the nodes involved in the operation share their parents after the reversal. This process is outlined in Figure 2.5. The operation is applied only between chance nodes and seeks making a chance node suitable to be deleted.

With the operations described above, the algorithm can be performed. Once the model is ready for the inference (the "no forgetting" arcs are added and the barren nodes are eliminated), the algorithm of Tatman & Shachter (see Algorithm 2.2) goes into an iterative process to select which node(s) will be deleted. The algorithm is quite similar to the original one, as it just add some operations to deal with Super Value Nodes. The first line of the iteration process deals with them indeed. In the event that two value nodes (V_1 and V_2) have the same successor (V_3) it has to be, by definition, a super value node. If the set of conditional predecessors of V_1 is a subset of the one of V_2 and they are the only informational predecessors of V_3, they can be removed. If the SVN has more informational predecessors, V_1 and V_2 can be merged.

The rest of the iterative process, deals with the selection of a decision or a chance node and its elimination. First, the algorithm tries to find an eligible decision node. There are two scenarios for eligibility in the case of decision nodes. The first one, when the decision node has among its successors just one value node and it has to happen also

Algorithm 2.2 Arc reversal for models with Super Value Nodes algorithm.

```
1: add "no forgetting" arcs
2: eliminate all barren nodes
3: while value nodes have conditional predecessors do
4:    if there is any set of value nodes that fit the subset rule then
5:       remove these value nodes (with a sum of a product of them)
6:    else
7:       if there is a removable decision node then
8:          remove the decision node and the necessary value nodes
9:          eliminate any resulting barren nodes
10:      else
11:         there must be a removable chance node that will be removed and the necessary
            value nodes (maybe some arcs must be reversed)
```

Algorithm 2.3 Arc reversal iteration.

```
1: while the chance node has successors do
2:    find a chance node among its successors with no other directed path between them
3:    reverse this arc
4: remove chance node
```

that all the conditional predecessors of the value node, besides the decision node, are also informational predecessors of the decision node.

If among the successors there are several but only values nodes, there are two more conditions to be met. But we need two more definitions. The indirect predecessors of a node are all nodes in the diagram that belong to a directed path that leads to that node. The terminal decision node in a graph with super value nodes is the one that agglutinates all of them and has no successors. With these definitions, we can explain the conditions needed in this second case. The first one, is that there is a value node (V_s) which ancestors are a subset of the set formed by the informational predecessors of the decision node and the decision node itself. The second one, that all the directed paths from the decision node to the terminal value node of the graph contain V_s. But before removing the decision node, those value nodes that belong to the indirect predecessors of V_s ($C_{ind}(V_s)$) are removed and then V_s is maximised over the decision node.

Should no decision node be eligible for deletion, a chance node will be. If there is a chance node which only successor set contains value nodes—it can be only one node—, it can be deleted. If it has more than one value node as successors, these are merged. Then the resulting value node inherits the conditional predecessors of the chance node. If there is not a chance node with these characteristics, we look in the ID for a chance node that has no decision nodes among it successors. Then, Algorithm 2.3 is applied in order to get

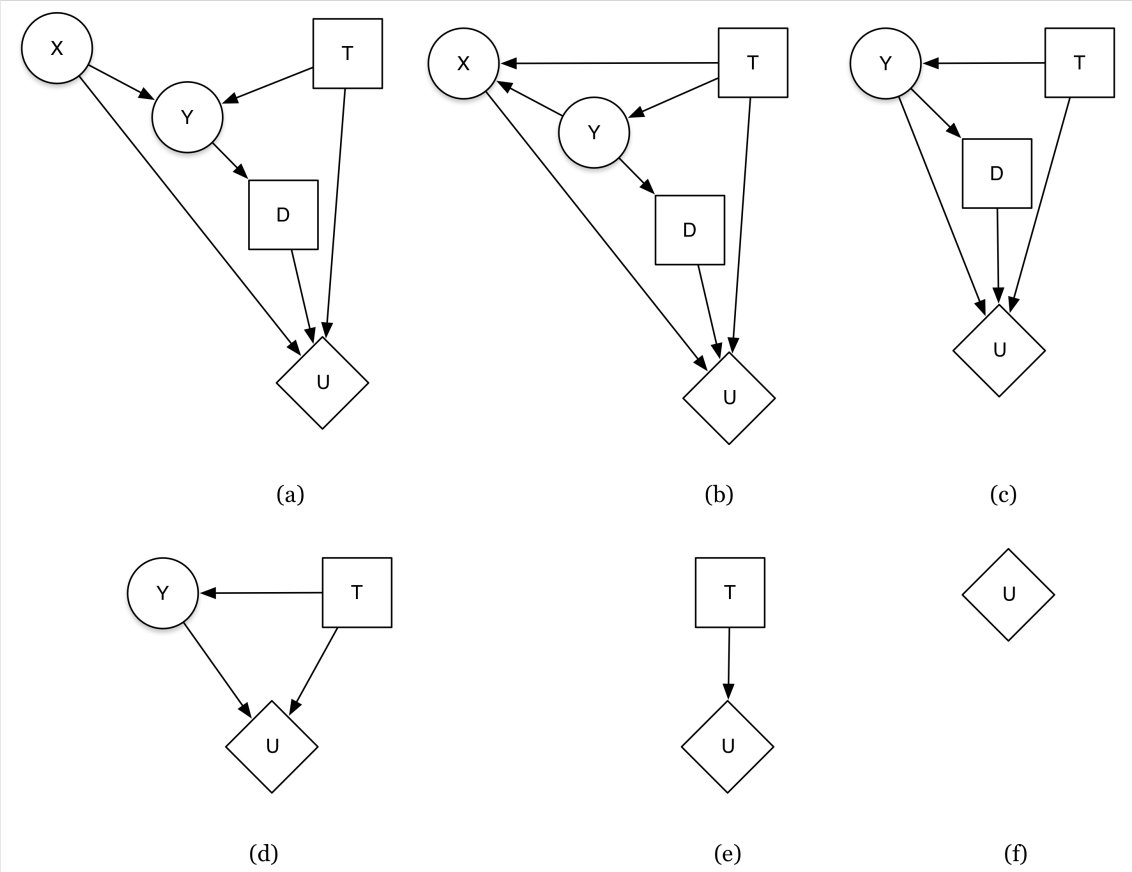

Figure 2.6: An example of the Variable Elimination inference. The figure (a) is an influence diagram. The figure (b) represents the arc reversal between X and Y to be able to remove X, that is seen in figure (c). The remaining steps are the same as in Variable Elimination.

a chance node with only value nodes among its successors and then it can be removed. Basically these operations make a decision node to met the conditions explained before where the chance node was eliminated directly without any extra steps.

In Figure 2.6 it can be found an example of the arc reversal algorithm applied to the same network as in the example of the variable elimination algorithm.

2.4.3 Strong junction trees

The strong junction tree algorithm (Jensen et al., 1994), as the variable elimination algorithm uses a strong variable elimination order (see Subsection 2.4.1), but opposite to variable elimination and arc reversal algorithms, it relies on an auxiliary structure—a junction tree—where all the computation is done. This structure is not novel of the article where the algorithm is outlined. The contributions of the authors focus on the way

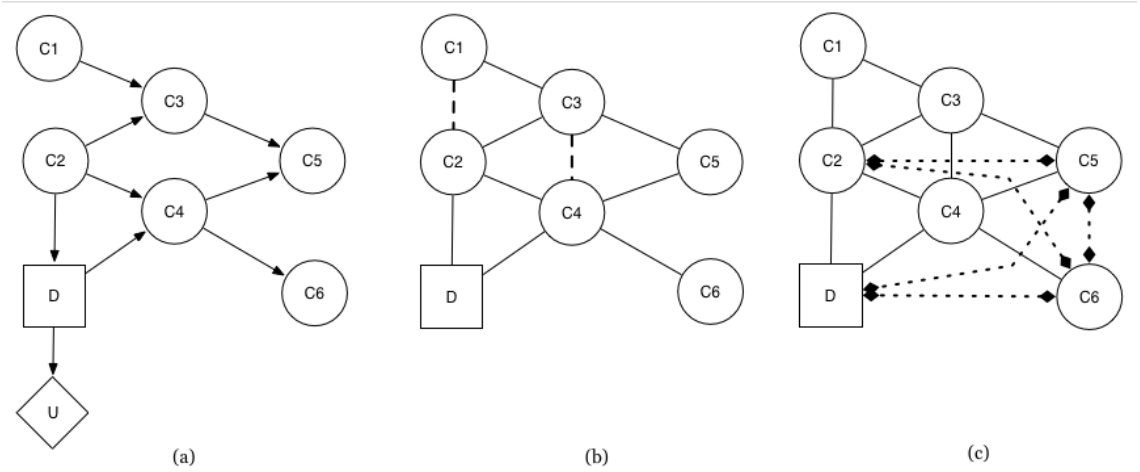

Figure 2.7: An example of the moralisation and triangulation process inference. The figure (a) is an influence diagram. The figure (b) represents the moral graph, where the links to marry the nodes appear with a dash line. Figure (c) shows the triangulation links. The figure in an adaptation from Jensen et al. (1994).

the junction tree is built and the way the messages are passed along it.

In a junction tree each *clique* (each node of the tree) represents a set of nodes of the Probabilistic Graphical Model. Should a variable of the network be present in two different *cliques* it will belong also to any group of nodes in the path that connects both *cliques*; this is called the junction tree property. Finally, each *clique* will have a potential with two lists of potentials, a probability list and a utility list. Each potential in the model will be put in the corresponding list of a clique that contains all its variables.

So, the first step to create the strong junction tree is to create a moral graph (Lauritzen & Spiegelhalter, 1988). This graph modifies the PGM linking—if they were not already connected—those nodes that share a child; and then losing all link orientation in the graph. Then, the utility nodes are drooped from the influence diagram. Strong elimination order plays a role in the next step. The obtained moral graph is then triangulated, if is not triangulated already. Losing the directions in the links, the model is no longer acyclic. Any graph is triangulated if for any cycle longer than three there is at least one link between to non consecutive nodes in the cycle. The graph is went through in the reverse order indicated by the strong variable elimination order. And if any cycle that not complies with the requirements is found, a link on it is drawn. At the end, a *strong* triangulated graph is obtained. An example of this process can be seen in Figure 2.7.

Finally, following the same special elimination order used for the triangulation, the *cliques* are created. The root of the strong junction tree is named strong root and all the links are directed towards the leaves. Between two *cliques* there is a *separator* that is

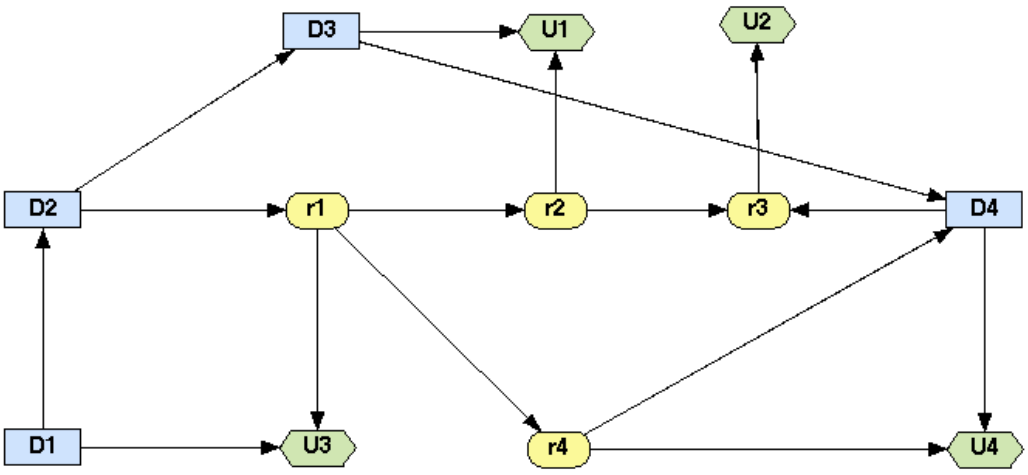

Figure 2.8: An influence diagram borrowed from Nilsson & Lauritzen (2000).

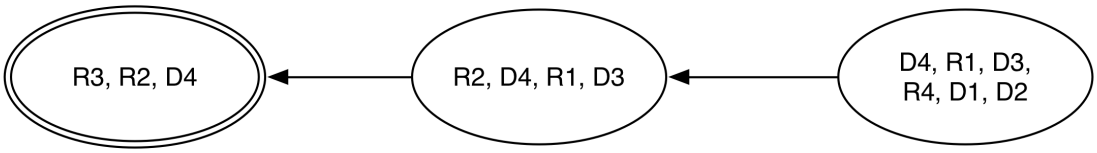

Figure 2.9: The strong junction tree of Figure 2.8.

formed by the variables present in both groups.

In Figure 2.8 it can be seen an ID borrowed from Nilsson & Lauritzen (2000) and modelled in OpenMarkov and in Figure 2.9 it is drawn the strong junction tree associated to the model.

When the strong junction tree is built, all the potentials in the *cliques* are merged. The probability potentials are multiplied between them and the utility potentials are summed. Then the messages from the leafs towards the strong root go along all the way up. When the potentials visit a *clique* its potentials absorb the potentials of the message and the new message continues its way to the strong root. Having two cliques, C_1 and C_2, and the separator S in the middle of them, and being their probability potentials ϕ and their utility potentials ψ, the message from C_2 is absorbed by C_1 as follows:

$$\phi'_{C_1} = \phi_{C_1}\phi_S$$

$$\psi'_{C_1} = \psi_{C_1} + \frac{\phi_S}{\phi_S}$$

$$\phi_S = \sum_{C_2 \setminus S} \phi_{C_2}$$

$$\psi_S = \sum_{C_2 \setminus S} \phi_{C_2} \psi_{C_2}$$

The $\sum$ operator is a maximisation or a marginalisation, depending on the variable that is being eliminated, a decision or a random node, respectively. Also of note is that the variables are eliminated with the same order used for the triangulation. The strategy is created with the maximisation of each decision that belongs to the *clique* closest to the strong root. When the message(s) arrive the strong root and the last eliminations are performed, the goal is reached.

2.4.4 LIMID conversion

The LImited Memory Influence Diagrams or LIMIDs were introduced by Lauritzen & Nilsson (1999). This model takes the basics of IDs and relaxes the two assumptions that we described in the opening of this section: the decision ordering and the no-forgetting assumptions. In general, finding optimal strategies within LIMIDs is prohibitive and thus usually approximate algorithms are applied to solve them. However, there exist some LIMIDs that are soluble and for them and exact solution can be calculated. A LIMID is soluble if all its decisions are extremal, and a decision is extremal if it fulfils the following assertion about d-separation:

$$u \perp_{\mathcal{L}} (\bigcup \{fa(d) : d \epsilon \Delta \setminus \{d_0\}\}) | fa(d_0) \tag{2.4}$$

This equation, in words, designate as extremal decision node (d_0) a decision node that is d-separated from every utility node that is among its descendants, from all the families (the union of a node and its parents) of the decisions in the LIMID but the family of the decision d_0 given the family of this decision. When a decision node is marked as extremal, it is converted in a chance node and a new extremal decision is tried to be found. If every decision node in the LIMID is extremal, the model is soluble.

If we draw in a model links from the family of a decision to every decision that succeeds it in the temporal order, the influence diagram will continue to be faithful with the assumptions indicated above. This new model is called the LIMID version of an ID. These models are soluble and therefore exact solutions can be found. In soluble LIMIDs there is an optimisation available. It is based in eliminating non-requisite links from

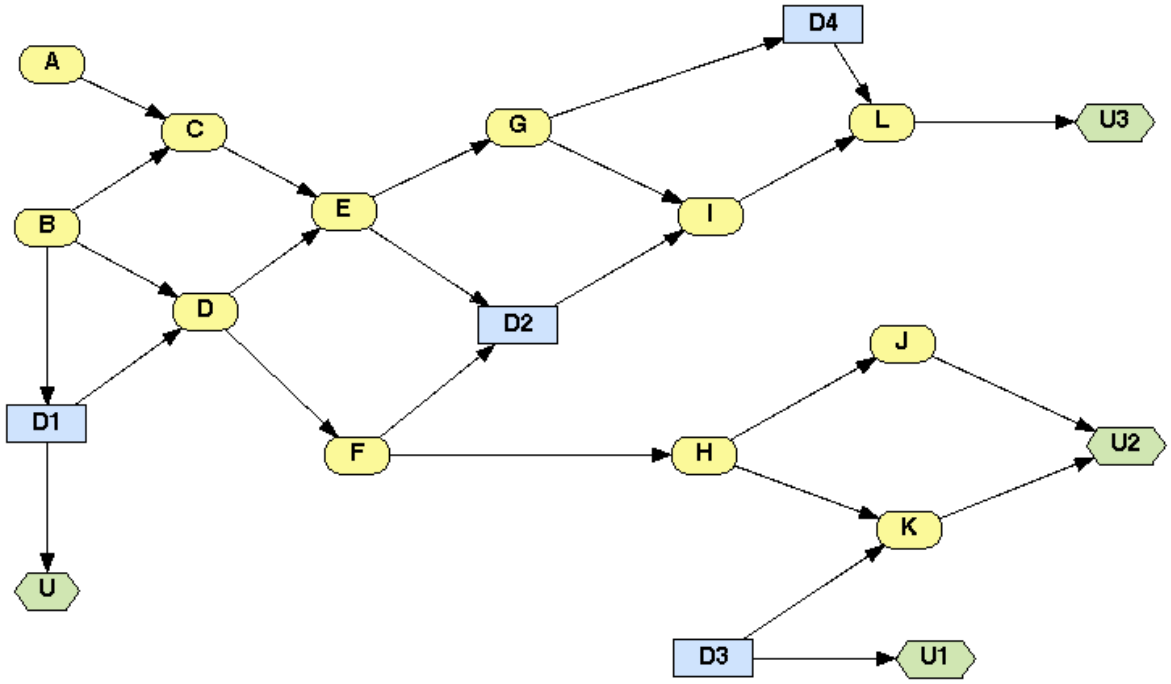

Figure 2.10: An ID from Nilsson & Lauritzen (2000).

parents of the decision nodes to their children that are of decision type. This way, some links may be eliminated and fewer calculations may be performed. This model is called the minimal reduction of the LIMID. The cost of the reduction is linear $O(k(grap\_size))$.

As the strong junction tree, the algorithm proposed needs an auxiliary junction tree structure but the order to build it is not constrained by the temporal order in the graph and thus is simply called junction tree. Both this and the strong junction tree algorithm are not the only ones that use this structures, but each algorithm modifies the creation or the way the messages are passed.

For the LIMID conversion algorithm, the order for the triangulation of the moral graph is established by selecting the node that would create the smallest *clique* in the tree. The objective behind this design is to create small *cliques* and hence manage smaller potentials that in the end would need fewer computations efforts—but this set-up comes with a cost in the message passing as we will explain later in Chapter 3. Both the junction tree and the strong junction tree of Figure 2.8 can be seen in Figures 2.13 and 2.14. Before making any other operations, each *clique* potentials are initialised as explained in the strong junction tree.

In Figure 2.13 the links of the tree have no directions. This is because the authors propose the use of the Single Policy Updating algorithm to perform the inference on it.

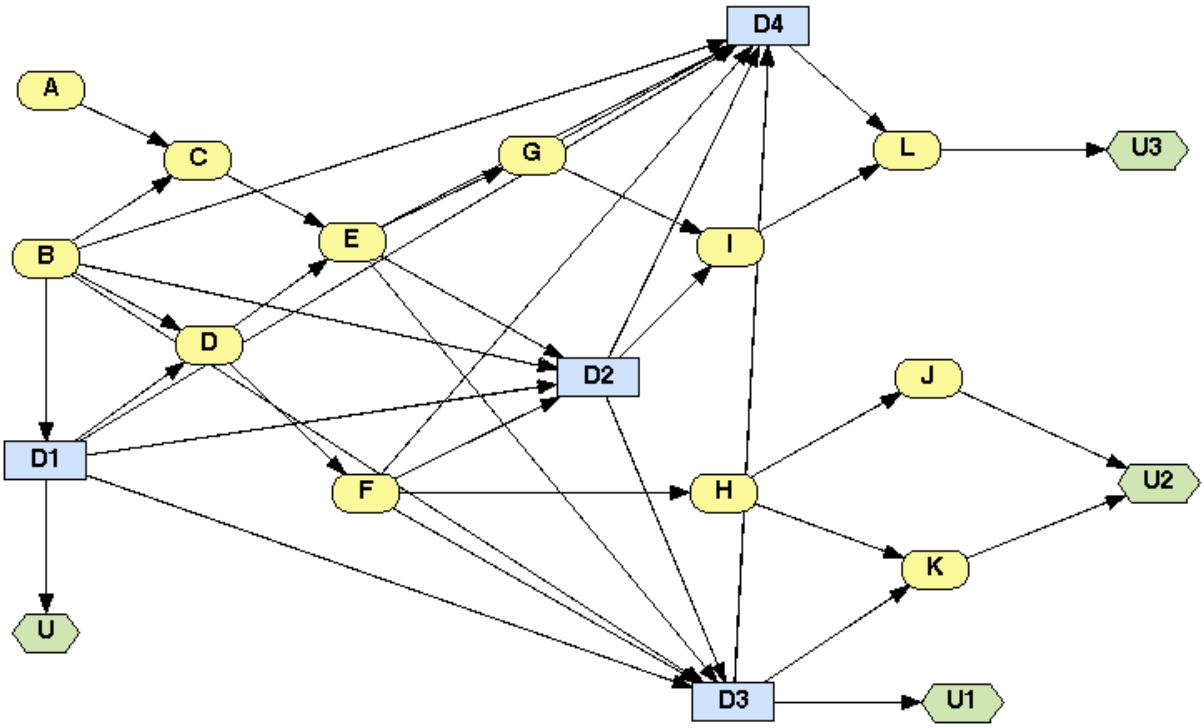

Figure 2.11: The LIMID version of the ID of Figure 2.10.

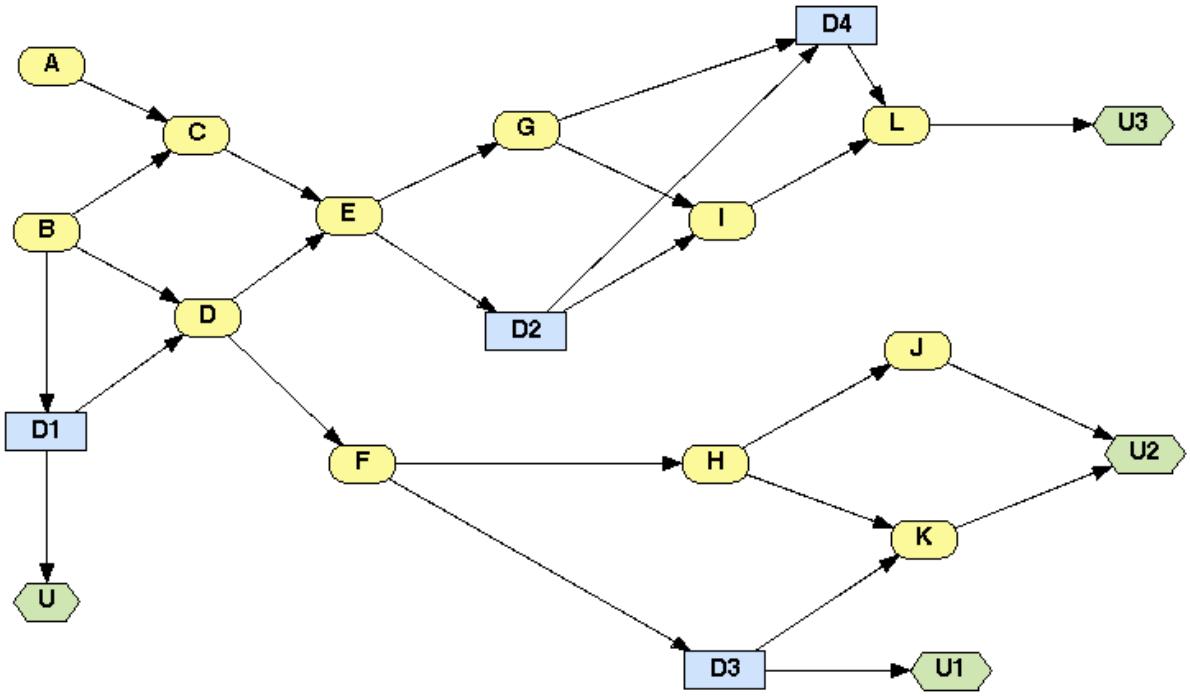

Figure 2.12: The minimal reduction of the ID of Figure 2.11.

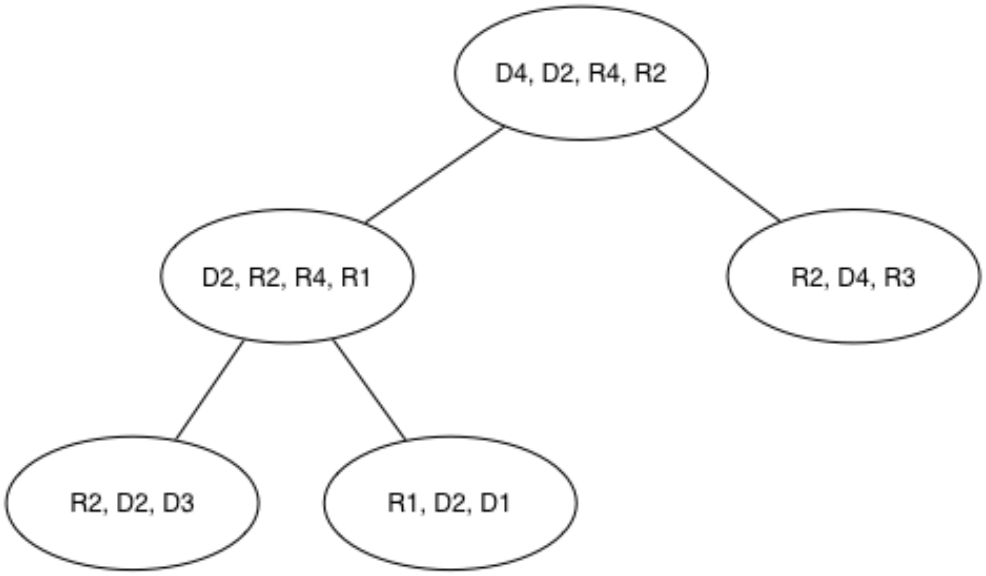

Figure 2.13: Junction tree of the minimal LIMID of Figure 2.8.

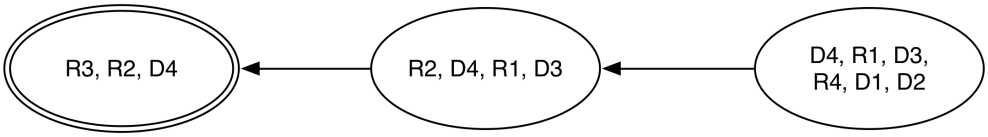

Figure 2.14: Strong junction tree of Figure 2.8.

The Algorithm 2.4 selects a decision node and then searches for a *clique* that contains the family of the decision node. This node will act as root of the junction tree and all the messages will be directed towards it. Once the messages reach the root, the calculated optimal policy for the decision is multiplied in the probability potential of the clique and then the process continues till all the decisions have been taking in count. At the end of this process, both the optimal strategy and the maximum expected utility are calculated.

So it is clear that more messages are passed in this algorithm than in the strong junction tree. Nevertheless, the authors propose two improvements to the algorithm that involve fewer messages moving in the tree.

The first improvement states that not all the messages calculated in an iteration after the first one. Only the messages in the path from the new root to the old root need to

Algorithm 2.4 Single Policy Updating.

1: **for all** $d_i \in [k..1]$being $d_k..d_1$ the inverse of the exact ordering solution of the decisions **do**
2: Calculate the optimal policy for d_i

be calculated. The previous messages can be merged with the new ones to calculate the optimal policy of the iteration. Lauritzen & Nilsson call this improvement partial collect propagation.

The second improvement says that some messages are not needed and thus, some branches of the tree do not have to be visited in certain iterations. When a *clique* acting as root (R) has a neighbour (C), and their intersection is a proper subset of the parents of the decision node which optimal policy is being calculated, this optimum policy can be computed without the message from C. This way, it is possible to avoid all the passing of messages to C as well.

2.5 State of the art of the comparison of the algorithms

There are in the literature few empirical comparisons of the performance of the algorithms described in Section 2.4. We have been able to track just one paper (Luque & Díez, 2010) where the authors compare and contrast the time and memory consumed by hundreds of networks which inference was performed using variable elimination and arc reversal. In the paper by Butz et al. (2009b) the authors make experiments with six large Bayesian Networks. These two studies obtained that VE is never slower than AR and that the former can even be significantly faster than the latter. There is another study (Butz et al., 2009a) that compares the lazy variations of these algorithms. In (Madsen & Nilsson, 2001) the authors make a comparison between some algorithms (none of the ones in the scope of this dissertation) but they only depict some information about the number of operations involved in the different inferences. To the best of our knowledge this is a novel empirical comparison of all the algorithms referred in the previous section.

Chapter 3

Empirical comparison of the algorithms

3.1 Introduction

As outlined in Chapter 1, previous works (Luque & Díez, 2010; Butz et al., 2009b) have carried out experiments which results indicated that variable elimination requires less memory and is faster than arc reversal. To the best of our knowledge, no other studies have confronted empirically the variable elimination, arc reversal, strong junction tree and the LIMID-conversion algorithms. Nevertheless, *a priori*, strong junction tree and conversion to LIMID should perform worse than variable elimination regarding the speed, as neither of the formers eliminate redundant variables; variable elimination is capable of detecting when a variable is not necessary to compute an optimum policy and therefore is able of ignoring it. Also, strong junction tree and LIMID-conversion algorithms need to create intermediate structures to perform the inference. The results of our tests are in Section 3.4.

3.2 Implementation of the algorithms

3.2.1 OpenMarkov

OpenMarkov[1] (Bermejo et al., 2012), an Open Source tool for Probabilistic Graphical Models, that was born as Carmen (Arias & Díez, 2008) an renamed in 2010, started in 2002 at the Department of Artificial Intelligence of the Universidad Nacional de Educación

[1] `http://www.openmarkov.org`

a Distancia (UNED), in Madrid, Spain. The project commenced with the knowledge acquired in the construction of Elvira[2]. In pursuance of multi-platform interoperability, the tool is programmed in Java. OpenMarkov is able to represent several types of networks as well as several types of temporal modes and all these are stored in ProbModelXML formats—see Arias et al. (2011) for definitions and references. Though it can evaluate only a subset of all the networks it is able to represent, the number of network types which inference the tool is capable of handle is growing. There are three types of variables in OpenMarkov: finite-states, numerical, and discretized. This last type has a finite set of states, each one having an associated numeric interval.

In this master thesis, three algorithms have been incorporated to OpenMarkov: arc reversal, strong junction tree and conversion to LIMID. The algorithms developed can handle Probabilistic Graphical Models with several value nodes, which none of them can have children—so no Super Value Nodes (SVNs) are allowed in the networks. We will discuss some characteristics of their implementations in the remaining sections of this chapter.

3.2.2 Arc reversal

The original algorithm proposed by Jensen et al. (1994) is able to manage networks with just one value node. The algorithm was extended by Tatman & Shachter (1990) to cope with SVNs; however, the algorithm only accepts non-negative numerical values. We have developed a hybrid algorithm that manages several value nodes with negative numerical values, if necessary. Our implementation is very similar to the original algorithm and takes some notions from the newest one.

3.2.3 Strong junction trees

OpenMarkov has an algorithm to evaluate Bayesian networks with junction trees. The code of the algorithm has been adapted to handle also influence diagrams. The code we have developed does not need to triangulate the moral graph as a previous step of the construction of the strong junction tree, and the creation of the cliques is done directly from the moral graph. This is an improvement compared to the original algorithm. Actually, the output of the algorithm in OpenMarkov is a forest, not a tree, because the same networks can be expressed with different trees.

[2] `www.ia.uned.es/~elvira`

3.2.4 Conversion to LIMIDs

In the implementation of the algorithm proposed in Nilsson & Lauritzen (2000) the authors did not state the heuristic used to select the node that would create the smallest clique. There are two classic metrics: the number of variables (Kjærulff, 1993) and the combined number of the states of the variables that the *clique* will host (Huang & Darwiche, 1996). We have chosen a hybrid solution and we have taken both metrics. We apply the first one and if a tie is produced, we apply the second metric. The auxiliary structure used by this algorithm is also a junction tree, like in strong junction trees, but it has several differences with this model that have affected the implementation. To start with, the junction tree of the LIMID-conversion has no directions in the links of the *cliques*, as several of them may act as root during the Single Policy Updating process. It is also of note that in this case the messages that move along the tree need to be stored to avoid calculating them more than once, as several iterations are performed.

3.3 Design of the experiments

3.3.1 Generation of the influence diagrams

We used two different kinds of influence diagram to test the algorithms. We programmed in OpenMarkov the generation of both networks, to which we will refer to as the *n-test medical decision problem* and the *follow-up problem*, respectively.

3.3.1.1 The n-test medical decision problem

Two examples of this network can be seen in Figures 3.1 and 3.2. The network follows a pattern, and if the first figure shows the smallest network of this kind, the second one shows the fourth iteration of the network. While chance variable X represents a disease, a non-observable variable, the decision variable Tr corresponds to a treatment. The states of the disease node can be either *present* or *absent* and their values are settled with the prevalence of the disease, that gives out the proportion of a population that may have a certain condition. If the value of the *present* state is equal to the prevalence, the value of the *absent* state is $1 - present$.

The Tr node has a domain of *yes* and *no*—to express whether the treatment is applied or not—and is linked to a utility node, U_0, that reveals the Quality of life associated to the treatment. Then, to complete the example network, there is a set of three nodes: a decision node (T_1) that has two children, a chance node (R_1) and a utility node (U_1). This set

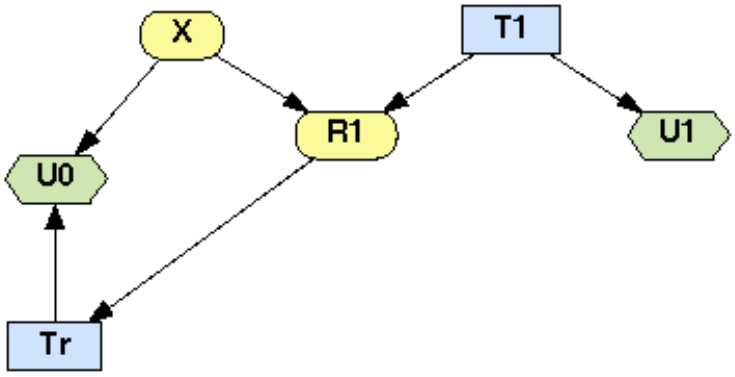

Figure 3.1: The n-test medical decision problem with 1 test ($slice = 1$).

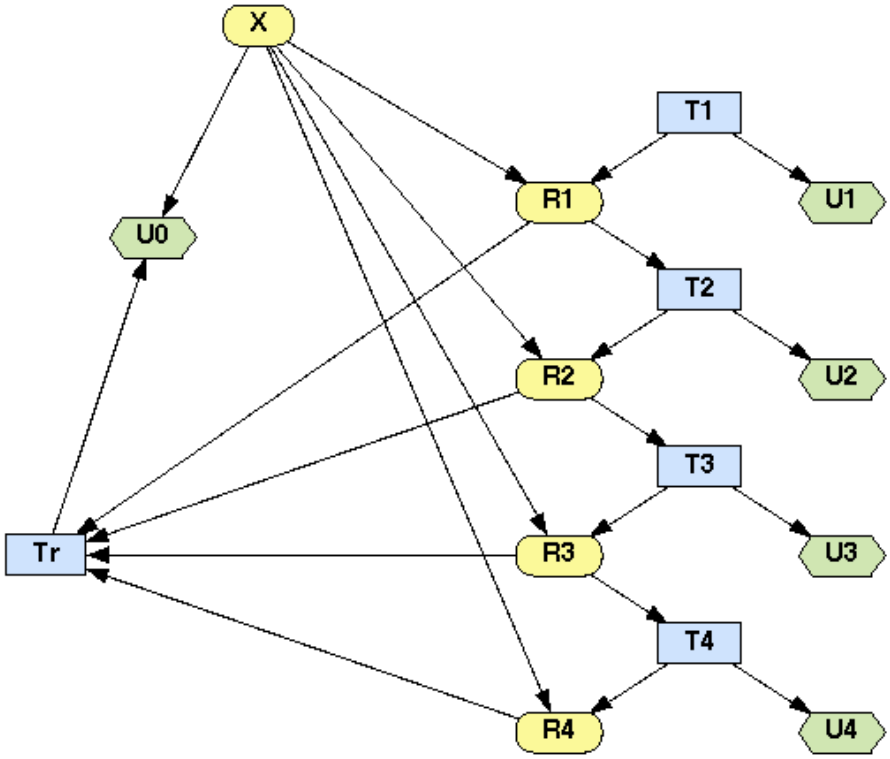

Figure 3.2: The n-test medical decision problem with 4 tests ($slice = 4$).

stands out for the decision of making a test (with the same domain as the treatment), the results obtained (the node R_1 has three possible states: *positive*, *negative* and *not_done*), and the medical cost-effectiveness of the tests. The characteristics of the test result are its sensitivity and specificity, that measure how well it performs; the sensitivity indicates the amount of positive outcomes correctly identified while the specificity does the same with the actual negatives outcomes obtained. The cost-effectiveness analysis is a resource in medicine to make decisions regarding health care interventions.

The links emerging from the disease node mean that the disease has en effect on the destination variables. The decision of applying or not a test changes its results and has a consequence in the cost, that is measured through the utility node related to it. Alongside, links headed to the treatment indicate knowledge at the disposal of the decision maker when the decision has to be made. Finally, the links headed to the U_0 node point to nodes that have an impact on the Quality of life.

To generate a network the user may call the method `generateIDTestProblem` in the OpenMarkov package `org.openmarkov.inference.tools`. The method requests four parameters that stand for the number of *slices* desired in the network, the prevalence of the disease, the minimum sensitivity and the minimum specificity. The *slices* of the network requested by the user determine the amount of test sets that the created network will have. In the case of the parameters related to the sensitivity and the specificity, these are referred as minimum because for every test in the network the method produces a random parameter between the specified values and 1, that then assigns to the potential of decision node accordingly. The values of the other variables in the network are completely random.

3.3.1.2 The follow-up problem

The follow-up problem, which we define in this thesis, consists of finding at each moment the best treatment for a patient. The ID built for this problem considers n time slices (n is called the *horizon*). In the i-th slice, the variable X_i, represents the presence or absence of a disease that cannot be observed directly. Whereas the observable variable Y_i stands for the presence or absence of a symptom. With D_i variable (it is also called the *action*) the decision maker chooses whether to apply or not a treatment to the patient—that may cure the disease or lessen the possibilities of it to reappear. Finally, U_i corresponds to the Quality of life of the patient in the time slice; it is affected both by the disease (or the status of the disease in subsequent time slices) and the decision taken. These nodes also have links headed to the following chance node, X_{i+1}, that is the sign of the evolution of the disease and the entry point of the following time slices that repeat the explained set. An example of this network with a horizon of 3 periods is reproduced in Figure 3.3.

This ID is a particular case of a Partially Observable Markov Decision Process (POMDP), in which we have an unobservable variable, X_i, and an observable one, Y_i. However, we will evaluate this model as a dynamic ID, i.e., we will obtain for each decision D_i the optimal policy as a function that depends explicitly on all the previous observations, $\{Y_0, \ldots, Y_i\}$, while in POMDPs, whose horizon is usually assumed to be infinite,

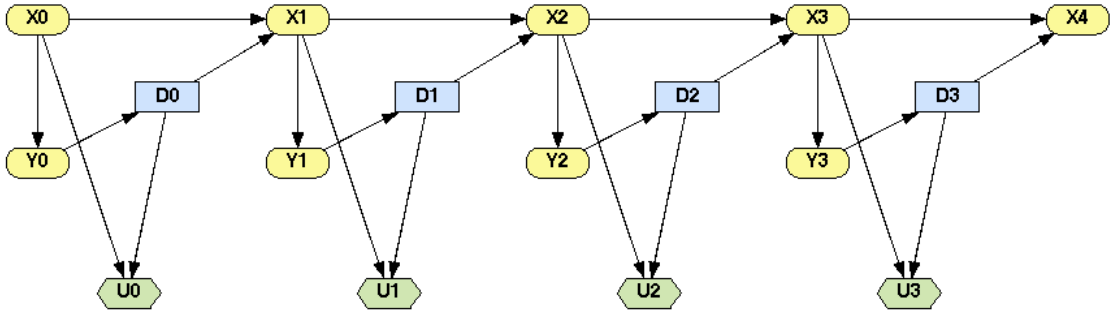

Figure 3.3: The follow-up problem, with a horizon of 3 periods ($n = 3$).

the optimal policy consists of a stationary policy that depends on the belief state (the probability) of the state variable, X_0 (Hansen, 1998).

To generate this kind of network, there is a method called `generateMedicalPOMDP` in the same package of OpenMarkov indicated above. The method shares two parameters with the method of the previous network, slices *(horizon = slices − 1)* and prevalence. It requests three more parameters, all related to the evolution of the disease. The first two are also minimum values to calculate a random value within a range of the received value and 1.

The `minAbsentNoTreatment` parameter reflects the minimum probability for the disease to be *absent* if in the previous time slice the state was also *absent* and no treatment has been applied. Should it have been applied, to calculate the probability of the same evolution and with the same previous states, the method adds a small amount (0.01) to the same value, as it is assumed that applying the treatment reduces the chances of having the disease. The second parameter, `minPresentNoTreatment`, is the minimum probability for the disease to be *present* if also the previous state was *present* and the treatment has been applied. Finally, the method takes the prevalence to calculate if the disease is *present* if it already was and no treatment has been applied. The last parameter `stationaryProbabilitiesAndUtilities` is for the user to indicate whether s/he wants stationary probabilities and utilities, what is common in POMDPs, or not. The values of the other variables in the network, the symptoms and the utilities, are completely random.

3.3.2 Experiments

Once we coded and tested the algorithms, we conducted an empirical study of their performance. We measured both the time and the memory the algorithms employ to perform the inference of the two kinds of influence diagram real-world applications described in

Subsection 3.3.1. We took the difference between two times measured using the precision timer `System.nanoTime()` of Java[3] to compare the first component of the performance and the size of the variables in memory for the second. Within each one of the problems described, we incremented the number of slices[4] till the feasibility border flagged by any of the algorithms not being capable of making the inference. The complexities we reached were 8 slices for the n-test medical decision problem ($n = 8$) and 9 slices ($horizon = 8$) for the follow-up problem. Therefore, we have tried the algorithms against 17 networks. While the follow-up problem is a multi-stage network, somehow *horizontal*, the n-test medical decision problem is somehow a *vertical* network, with two groups of nodes connected between them. Here we picture again both networks in the Figure 3.4 and the Figure 3.5 to illustrate this description.

We set up a database in `MySQL` (Widenius & Axmark, 2002) format to achieve some dynamic execution of the algorithms. The database has been used to store the basic configuration of the tests and hence if we needed to change certain parts of the experiments we did not need to change the source code. We were able to know which networks had been generated, to configure which networks were profiled, and the location of them, to name just some possibilities of the database.

Once we had the algorithms and the database ready, we compiled the sources in two executable `jar` format programs. One of them to test the efficacy of the algorithms and the other one to test their efficiency against the networks. The first program, the one that checks the inferences needs two parameters:

1. *ID type [string, one]* to specify the type of network that the user wants to test. For this research, MedicalDecisionProblem or MedicalPOMDP.

2. *InferenceAlgorithm [string, one]* to specify the inference algorithm that will be used. For this research, VariableElimination, ArcReversal, StrongJunctionTree or LIMIDversionID.

The program will take all the networks that had been generated of the type received and will test their efficacy against variable elimination, if it has not be done before. All the algorithms have to obtain the same result in the inference than variable elimination. If this is not the case, the algorithm will be stored in a table of the database that indicates that an error has been produced. If the algorithm in test is arc reversal, the elimination

[3] `http://docs.oracle.com/javase/7/docs/api/java/lang/System.html`, checked the 6th of September, 2014

[4] In the rest of the chapter, the slice term matches n in the case of the n-test medical decision problem and $horizon + 1$ in the follow-up problem

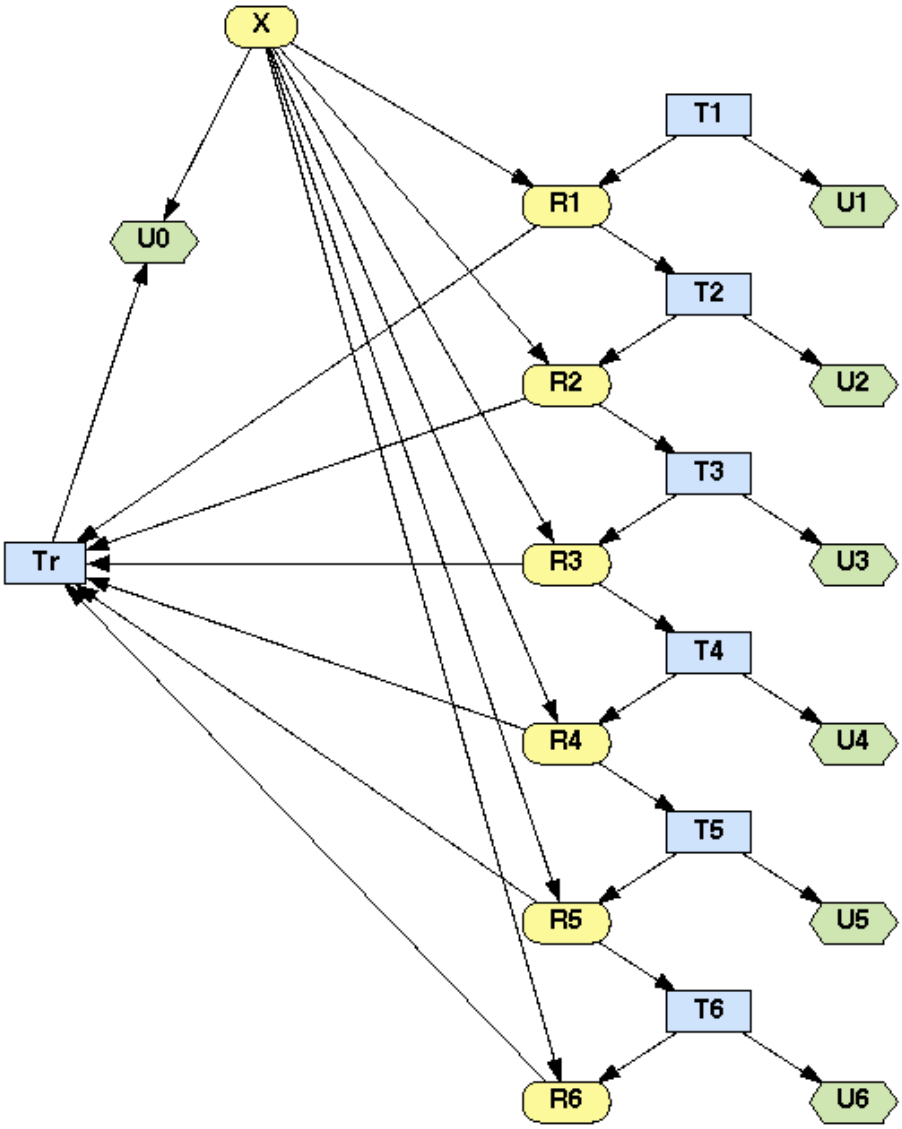

Figure 3.4: n-test medical decision problem with 6 tests ($slice = 6$).

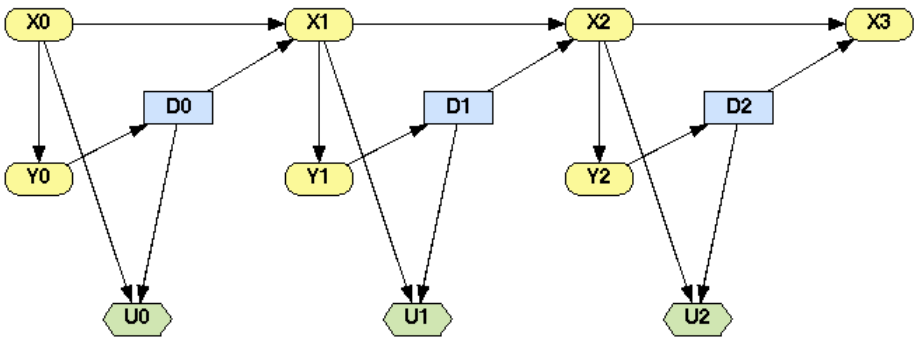

Figure 3.5: The follow-up problem, with a horizon of 2 periods ($n = 2$).

order of the algorithm is stored, to force variable elimination to use it—further detail in Section 3.5.

The second program retrieves the data—time and/or space use—from the different inferences and needs the following parameters:

1. *db [true or false]* to store or not the obtained information in the database
2. *csv [true or false]* to store or not the obtained information in `csv` files
3. *forceGarbageCollector [true or false]* to force or not the Java Garbage Collector in several points of code
4. *measureTime [true or false]* to measure or not how much time does it take to an algorithm to perform the inferences
5. *measureSize [true or false]* to measure or not how much memory does an algorithm need to perform the inferences
6. *forceVariableEliminationOrder [true or false]*, to force or not in variable elimination algorithm the same elimination order than in arc reversal
7. *numberOfIterations [integer]* if measureSize is true, the number received will be disposed and only one iteration will be made
8. *deletePreviousProfiling [true or false]* to delete or not the previous tests. If they are not deleted, only those experiments that are missing from the database are performed.
9. *interventionsOff [true or false]* to take in count or not the time used by variable elimination during the calculation of the interventions.
10. *ID type [string, one or more]* to specify the network type that the user wants to test. For this research, MedicalDecisionProblem or MedicalPOMDP.
11. *InferenceAlgorithm [string, one or more]* the 10^{th} and subsequent parameters are the inference algorithm(s) that will be used. For this research, VariableElimination, ArcReversal, StrongJunctionTree and LIMIDversionID.

When we carried out the collection of the samples with the same machine (see Table 3.12) we did it in several stages; we also needed several attempts to circumvents some problems that will discuss later in this Section. In any case, the time and size were

measured independently for the experiments not to collide one with each other. We also did our best to shut down any non essential program or service in the machine, in order to interfere the less possible with the inferences. All the data was stored in the MySQL database and we then analysed the collected information with the statistical program R (R Development Core Team, 2008). We created some R scripts to read the information from the database, for which also some views were stored in the database. Then we split the experiment's data, by kind of network and, then, by algorithm. Another script took this information and calculated some statistical values, like the mean and the median. All this information was used to generate some excel-sheets where the empirical comparison of the algorithms rests. Finally, we drew some plots to compare and contrast the algorithms graphically.

When designing the experiments, all the algorithms but variable elimination were allowed to play freely. For the purpose of comparing variable elimination and arc reversal under the same conditions, we imposed the variable elimination order in the first mentioned algorithm. The order elimination of each influence diagram we utilised in the study was established by the inference made through arc reversal. This process was done independently to the obtention of any efficiency data. Before exhibiting the results, we will discuss at the beginning of Section 3.4 some issues that we came across during the analysis of the performance of the algorithms and that lead to change some factors of the experiments.

3.4 Experimental results

Our firsts examinations involved conducting the inference of each algorithm over each network 100 times. We expected some extreme values because a computer is not a perfectly stable environment; for this reason we had decided to use the trimmed mean to juxtapose the time efficiency of the algorithms. This value is a robust statistical measure, that suits the analysis of heavy-tailed distributions (Stigler, 1973). It is associated with a numerical value that indicates the amount (%) of values of each tail that are removed. Hence, the trimmed mean 2 leaves out the 2% of the highest and lowest values of the data. When we pulled out the initial results, the values of the trimmed mean 2, 5, 10 and 15 were extremely different. Meaning that the distribution obtained was skewed. So we decided to trigger again the programs, but requesting 500 inferences over each network. The new results showed trimmed means that were nearly the same—the data we expected (see Table 3.1).

Figure 3.6: Configuration of the R environment used in the research.

```
## Loading required package:  methods
## Loading required package:  DBI
## Loading required package:  MASS
## Loading required package:  gld
## Loading required package:  mvtnorm
## Loading required package:  lattice
##
## Attaching package:  'PairedData'
##
## The following object is masked from 'package:RMySQL':
##
##    summary
##
## The following object is masked from 'package:DBI':
##
##    summary
##
## The following object is masked from 'package:base':
##
##    summary

## R version 3.1.1 (2014-07-10)
## Platform: x86_64-apple-darwin13.1.0 (64-bit)
##
## locale:
## [1] C
##
## attached base packages:
## [1] methods   stats     graphics  grDevices utils     datasets  base
##
## other attached packages:
##  [1] PairedData_1.0.1 lattice_0.20-29  mvtnorm_1.0-0    gld_2.2.1
##  [5] MASS_7.3-34      ggplot2_1.0.0    WriteXLS_3.5.0   data.table_1.9.2
##  [9] RMySQL_0.9-3     DBI_0.2-7        knitr_1.6
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.11.2      colorspace_1.2-4 digest_0.6.4     evaluate_0.5.5
##  [5] formatR_1.0      grid_3.1.1       gtable_0.1.2     munsell_0.4.2
##  [9] plyr_1.8.1       proto_0.3-10     reshape2_1.4     scales_0.2.4
## [13] stringr_0.6.2    tools_3.1.1

## Error:  package 'ggplot2' is required by 'PairedData' so will not be
detached
```

Table 3.1: Time employed ratio arc reversal / variable elimination in the n-test medical decision problem.

Slices	Mean	Median	TM2	TM5	TM10	TM15	Min	Max	Per5	Per95
1	0,89399	0,98807	0,98158	0,98674	0,98696	0,98468	0,97909	0,14906	1,05345	0,91645
2	1,07749	1,13645	1,11374	1,12181	1,12437	1,12259	1,07956	0,32511	1,18200	1,04394
3	1,22258	1,32117	1,29005	1,29679	1,30029	1,29814	1,24425	0,21570	1,37242	1,17867
4	1,74141	1,90437	1,83559	1,85673	1,87029	1,87202	1,78522	0,56922	1,94854	1,52076
5	3,50871	3,81281	3,64807	3,68389	3,71119	3,72712	3,46973	1,71780	3,83967	3,09235
6	8,50288	9,28006	8,99126	9,07203	9,12047	9,14873	8,28584	2,46212	8,97049	8,01920
7	16,73938	17,15829	16,57186	16,64021	16,74345	16,84370	15,28471	27,43599	15,87175	15,77993
8	22,91046	22,72740	22,89736	22,88547	22,85127	22,80890	22,43113	21,72129	23,61546	22,67344

Table 3.2: Time employed ratio arc reversal / strong junction Tree in the n-test medical decision problem.

Slices	Mean	Median	TM2	TM5	TM10	TM15	Min	Max	Per5	Per95
1	7,15187	7,39286	7,20266	7,22145	7,25342	7,28712	7,39474	4,88540	7,45122	6,54872
2	7,18187	7,15323	7,09377	7,07936	7,09145	7,10613	7,15455	5,51362	7,18967	7,03875
3	4,88787	4,72610	4,79393	4,77796	4,76916	4,75883	4,81356	14,52394	4,82443	4,96844
4	2,29240	2,20853	2,24693	2,24331	2,24153	2,23462	2,33295	5,88420	2,27427	2,23043
5	1,28133	1,23868	1,24525	1,24109	1,23990	1,23918	1,26587	4,21393	1,26214	1,26907
6	0,98807	0,97882	0,98013	0,97986	0,97860	0,97865	0,96869	2,96793	1,01074	0,97417
7	0,93911	0,91628	0,92018	0,92051	0,92104	0,92149	0,90929	3,91699	0,91489	0,92574
8	0,86580	0,86772	0,86893	0,86895	0,86905	0,86906	0,85838	0,88031	0,88977	0,84505

In spite of that, another values caught our attention; the inference of arc reversal was being made faster than the one of variable elimination. We examined the source of code of variable elimination and noted that within it, in every iteration, it was performed the search of the Interventions of the network; these are a tree structure that stores the optimal policy, but that are not needed for the inference itself. The process of their creation involves several loops, so we set off again the program against variable elimination without taking in count the time expended by the algorithm with the Interventions. We are in general confident about the consistency of the data, even though the final data produced still threw some unexpected results. We will continue this dissertation discussing them.

Figures 3.7 and 3.9 plot the trimmed mean at 2% against the slices of the networks for every algorithm we tested. This graphs, that have been obtained with the statistical program R (see configuration of the R environment in Table 3.6) show also the maximum and minimum values thrown by each algorithm in every slice, which illustrate that even

Table 3.3: Time employed ratio arc Reversal / LIMID-conversion in the n-test medical decision problem.

Slices	Mean	Median	TM2	TM5	TM10	TM15	Min	Max	Per5	Per95
1	4,16647	4,03247	4,14356	4,12353	4,10060	4,06873	4,16296	2,74161	4,39568	4,23315
2	3,02687	2,84295	2,96953	2,95242	2,94197	2,92812	2,95865	3,76862	3,17500	3,04296
3	1,26433	1,28808	1,34146	1,33646	1,33429	1,32584	1,34279	0,14542	1,43964	1,31406
4	0,44783	0,42666	0,44154	0,43955	0,43732	0,43478	0,43896	0,58749	0,47423	0,44838
5	0,18168	0,17632	0,17806	0,17776	0,17753	0,17738	0,17393	0,57457	0,18997	0,17416
6	0,11636	0,11615	0,11650	0,11656	0,11653	0,11648	0,12154	0,14434	0,12603	0,10866
7	0,09734	0,09635	0,09561	0,09569	0,09588	0,09607	0,09010	0,37099	0,09205	0,09637
8	0,08848	0,08848	0,08821	0,08825	0,08829	0,08833	0,08646	0,15562	0,08957	0,08632

using a high number of inferences there are some extreme values in the outcomes of the experiments. Figures 3.8 and 3.10 contain the same information, focused on those network of four slices or fewer.

We have divided the results into two sections, one per characteristic measured. The results, that we will detail later, *grosso modo* are:

1. Regarding the time expended in the inference by the algorithms:
   (a) In most cases, variable elimination performs better than arc reversal.
   (b) Strong junction tree beats the others in both types of networks till the fourth slice and in the follow-up problem even in the fifth one.
   (c) The LIMID-conversion, performs mostly like the strong junction tree till the networks with three slices but then its performance get worse exponentially
2. Regarding the memory employed in the inference by the algorithms
   (a) Generally speaking, variable elimination uses less memory space than arc reversal.
   (b) Strong junction tree is the best in almost all the cases.
   (c) The worst behave corresponds to the LIMID-conversion.

3.5 Analysis of the results

Next, we will dive on the details of the experiments, that are revealed in the tables of this chapter. In the tables, we present the ratio of the values of arc reversal over the

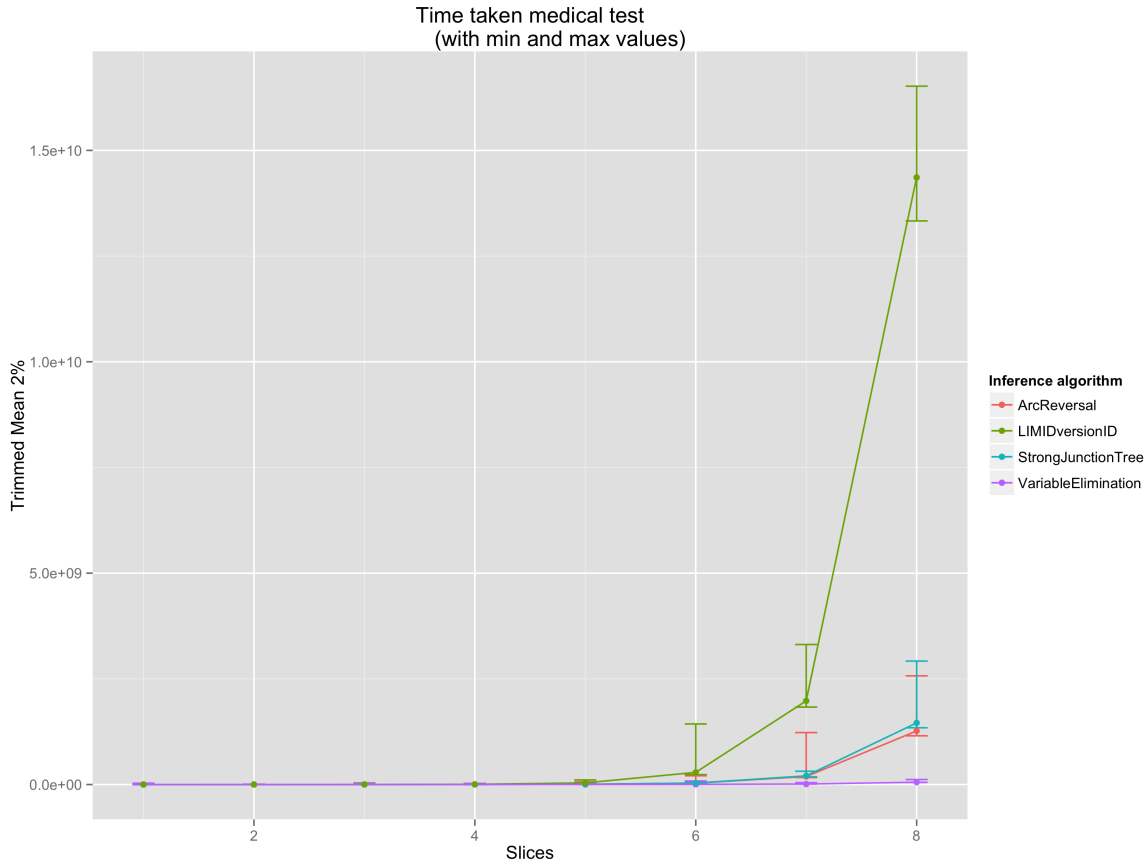

Figure 3.7: Trimmed mean 2 vs. slices in n-test medical decision problem inferences with minimum and maximum values.

other algorithms. A comparison of the four algorithms at the same time can be seen in the figures of the chapter. As we said before, variable elimination wins almost always to arc reversal both regarding the time utilised for the inference and the memory necessary during it. The ratios of the Tables 3.1 and 3.4 show that the time employed ratio grows exponentially with the slices of both kinds of networks. In these tables, though, arc reversal appears to be faster in the simplest networks. In the case of the n-test medical decision problem it is true that arc reversal has to eliminate one less node (X_1) than variable elimination because it becomes barren variable during the inference. But this is not the case in the follow-up problem network. We have not been able to find a reason for this value, although it could be due to the *forced* elimination order imposed to variable elimination. Maybe if it had performed freely, it would have been faster. Speaking about the memory occupied, only in the follow-up problem of 1 and 2 slices arc reversal need less memory than variable elimination. The reason is that arc reversal process one less node in each case because they become barren variables during the inference. It is true

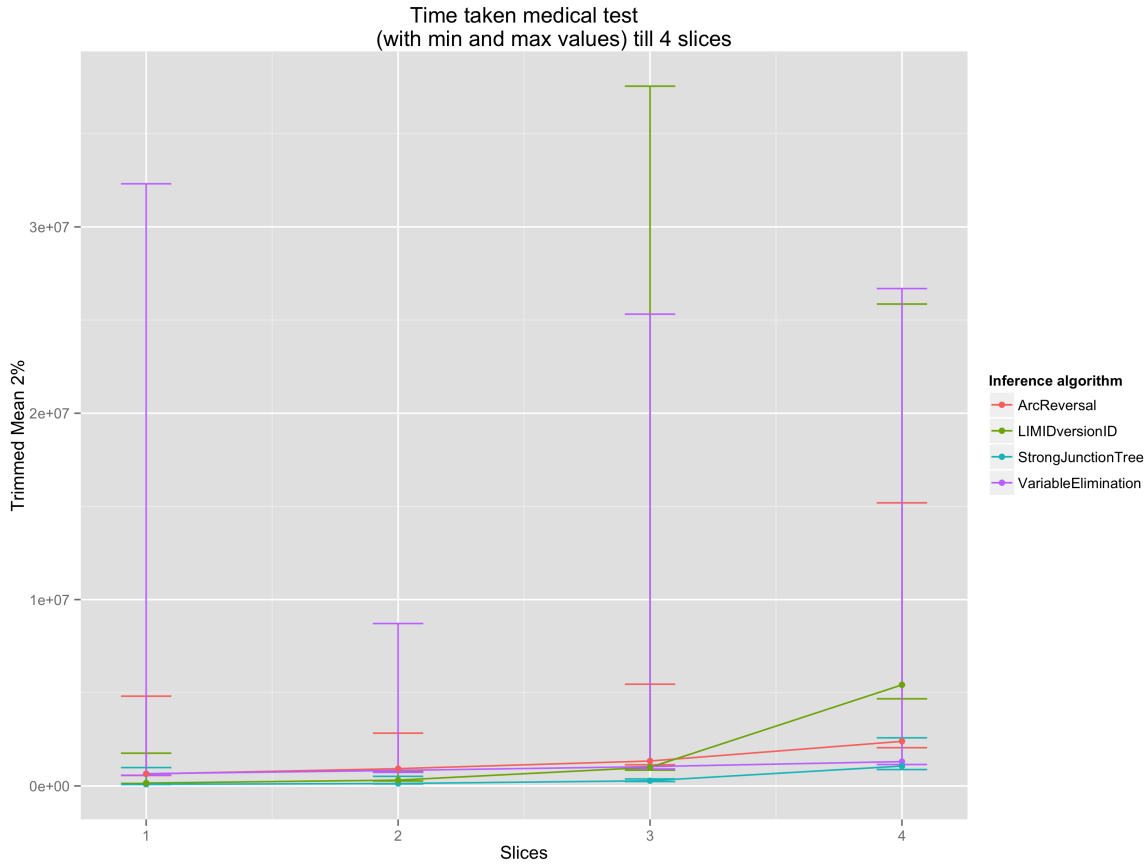

Figure 3.8: Trimmed mean 2 vs. slices in n-test medical decision problem inferences with minimum and maximum values, only of networks with four slices or less.

that in bigger networks arc reversal may remove up to four barren nodes that variable elimination has to process; but the elimination of these nodes also brings a cost, both regarding memory and time, as some operations have to be performed over the potentials. Bigger networks bring bigger potentials and thus these operations are more complex for both characteristics of the performance. Overall, we think the result about the speed in the simplest networks is a minor deficit in the research, that even could be related to the skewed distribution, and that out results match those of other authors and their analysis of these two algorithms.

Maybe the most surprising results are those obtained by the strong junction tree algorithm. The junction tree algorithms generalise the variable elimination algorithm and perform better than this, but only when querying the network about different probabilities. In the case of the variable elimination algorithm, when the inference about a certain probability is performed, and then the decision maker asks for another probability, the algorithm has not cached any other results. So it has to make all the calculations again.

Table 3.4: Time employed ratio arc reversal / variable elimination in the follow-up problem.

Slices	Mean	Median	TM2	TM5	TM10	TM15	Min	Max	Per5	Per95
1	0,78748	0,96610	0,93026	0,94380	0,95274	0,96031	1,00673	0,13077	1,00331	0,77446
2	1,19167	1,35043	1,27265	1,29525	1,32161	1,34278	1,38481	0,22300	1,37272	0,98437
3	1,39492	1,64984	1,53122	1,55606	1,58869	1,60831	1,58880	0,29950	1,60473	1,18779
4	1,68322	1,95435	1,82385	1,85966	1,89577	1,91772	1,80617	0,22161	1,87069	1,37247
5	2,06506	2,45069	2,30671	2,34649	2,38812	2,41465	2,19910	0,23812	2,28888	1,89819
6	3,01399	3,62021	3,52223	3,56938	3,61189	3,62478	3,15483	0,21413	3,26287	3,04281
7	5,10819	5,77563	5,79917	5,83803	5,88006	5,90035	4,97914	0,46631	5,12406	5,62426
8	10,95218	10,36900	11,26167	11,29245	11,27154	11,22331	9,22240	8,35552	9,26029	12,65420
9	19,48522	17,73383	19,78860	19,67054	19,50728	19,31561	15,91451	6,28657	15,89298	25,30379

Table 3.5: Time employed ratio arc reversal / strong junction tree in the follow-up problem.

Slices	Mean	Median	TM2	TM5	TM10	TM15	Min	Max	Per5	Per95
1	4,28569	4,32911	4,20340	4,25616	4,32043	4,29747	4,39706	6,62003	4,39130	3,19409
2	4,37582	4,68148	4,46764	4,52133	4,61523	4,62305	4,63559	1,02936	4,63292	3,44912
3	4,19439	4,32906	4,11885	4,16330	4,23986	4,24778	4,07426	11,20641	4,14098	3,24134
4	3,50237	3,66784	3,60802	3,63966	3,67945	3,69476	3,31536	0,54629	3,47200	3,07287
5	2,52438	2,57586	2,55941	2,56911	2,58450	2,59774	2,25490	0,80378	2,36449	2,43109
6	1,76373	1,69827	1,77262	1,77558	1,77663	1,77120	1,46233	1,42485	1,52512	1,87241
7	1,24633	1,17291	1,25164	1,24884	1,24451	1,23655	1,00643	2,18069	1,04191	1,45627
8	1,09179	0,95165	1,06782	1,06324	1,05727	1,04718	0,84363	5,21311	0,85452	1,31011
9	0,91495	0,82258	0,92793	0,92018	0,91072	0,89972	0,75091	0,26853	0,76241	1,20208

Table 3.6: Time employed ratio arc reversal / LIMID-conversion in the follow-up problem.

Slices	Mean	Median	TM2	TM5	TM10	TM15	Min	Max	Per5	Per95
1	3,55665	3,42000	3,52301	3,51813	3,48764	3,46636	3,28571	4,03175	3,29348	4,03320
2	3,08148	2,99526	3,05490	3,05963	3,06204	3,06579	2,92513	4,80514	2,91073	3,26394
3	2,51474	2,50123	2,50560	2,50993	2,51651	2,52214	2,31180	1,74158	2,35806	2,51373
4	1,61593	1,56563	1,61216	1,61429	1,62189	1,63504	1,40251	1,24845	1,47452	1,64183
5	0,70709	0,73245	0,75843	0,75724	0,75693	0,75789	0,63557	0,07627	0,66944	0,84035
6	0,34722	0,32817	0,34653	0,34551	0,34430	0,34266	0,27363	0,33995	0,28718	0,41551
7	0,18831	0,17630	0,18645	0,18502	0,18350	0,18224	0,14522	0,31819	0,14966	0,24537
8	0,13962	0,12156	0,13546	0,13429	0,13275	0,13131	0,10270	0,96516	0,10368	0,18683
9	0,10342	0,09136	0,10342	0,10229	0,10087	0,09959	0,07727	0,11526	0,07826	0,15432

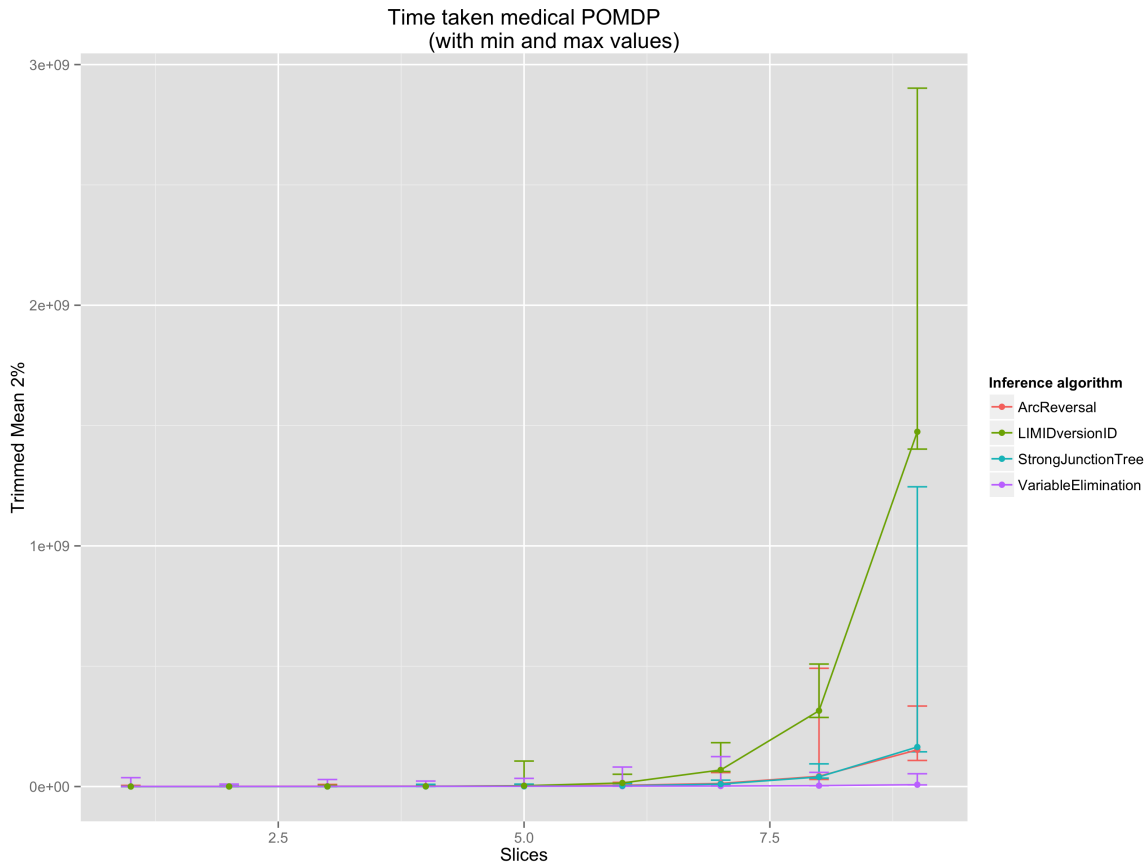

Figure 3.9: Trimmed mean 2 vs. slices in the follow-up problem inferences with minimum and maximum values.

As the junction tree can store, if desired, the intermediate calculations, later inferences on the network may be done faster at the expense of occupying memory. But in general is worth the expense. When only making one question to the network, the efficiency should be the same as the overall complexity of the two algorithms is the same. Given that we are just querying the network once, the question that arises is why strong junction tree performs better than variable elimination in 50% of the networks examples of the n-test medical decision problem tested and near the 56% in the case of the follow-up problem.

Before the creation of the clique tree, the algorithms chooses the elimination ordering in one single step while variable elimination invokes this subroutine each time a node is going to be deleted. This is not a trivial operation, as it consumes resources. There is another reason for strong junction tree being superior in the indicated cases and it is another advantage of the junction trees, and it is their ability to perform several operations over several potentials in a single clique. The cliques created in the different cases can be seen in Table 3.7. In spite of the advantages of strong junction tree over variable elimination

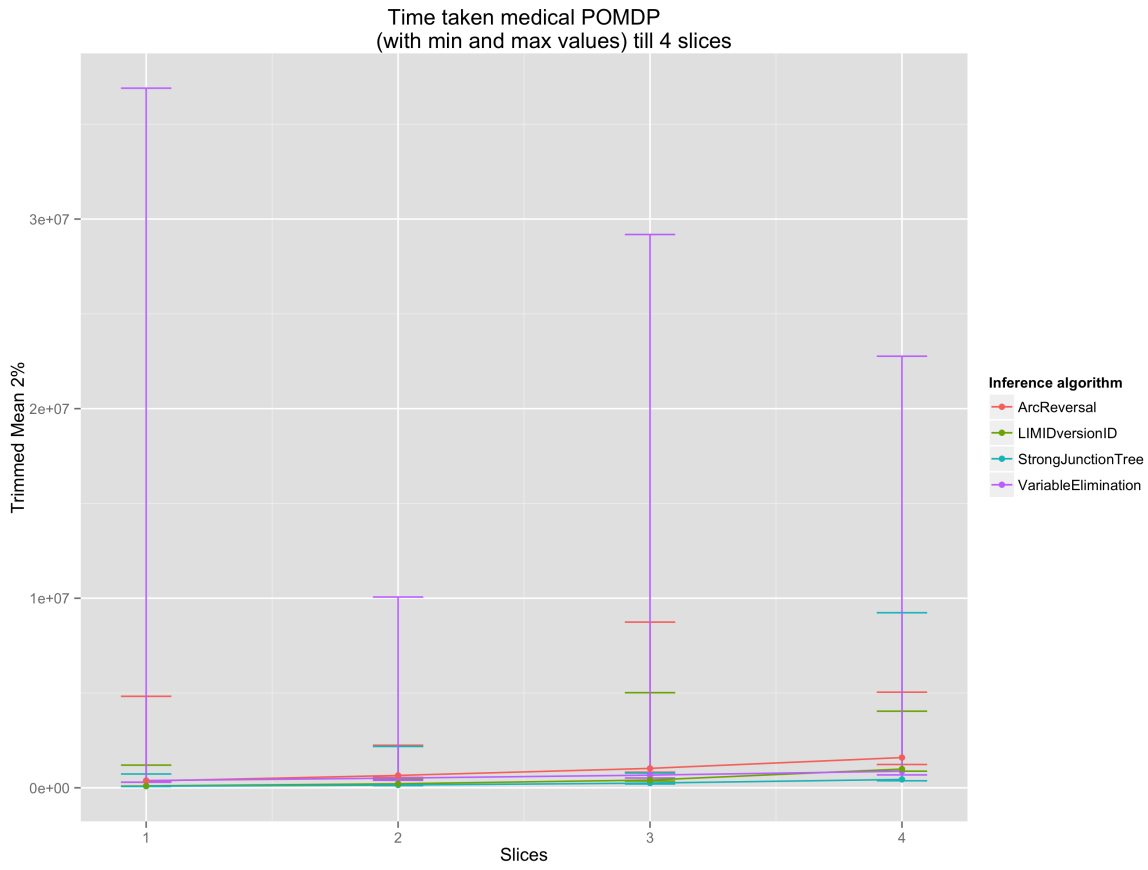

Figure 3.10: Trimmed mean 2 vs. slices in the follow-up problem inferences with minimum and maximum values, only of networks with four slices or less.

defined above, the complexity of the algorithm grows with the width of the clique tree. Therefore, as the size of the cliques scales, the performance of variable elimination retakes the leading.

In the tables that show the memory results of the inferences, we see opposite results than with the time efficiency. The numbers show that the worse the strong junction tree behaves regarding the time employed the better it behaves regarding the memory in use during the inference. It is the algorithm that uses the less amount of memory of all of them when run against the n-test medical decision problem. And this response is in almost 67% of the cases of the follow-up problem, where also the ratio soars. The reason for this performance is, once again, found in the cliques and their separators. Every time a clique sends a message upwards to the root, it eliminates all the nodes in the message that do not belong to the clique receiving the message. Variable elimination only removes one node per iteration. When there are some big cliques in the network, it is possible to eliminate several variables at once, and then, less memory is necessary. But these

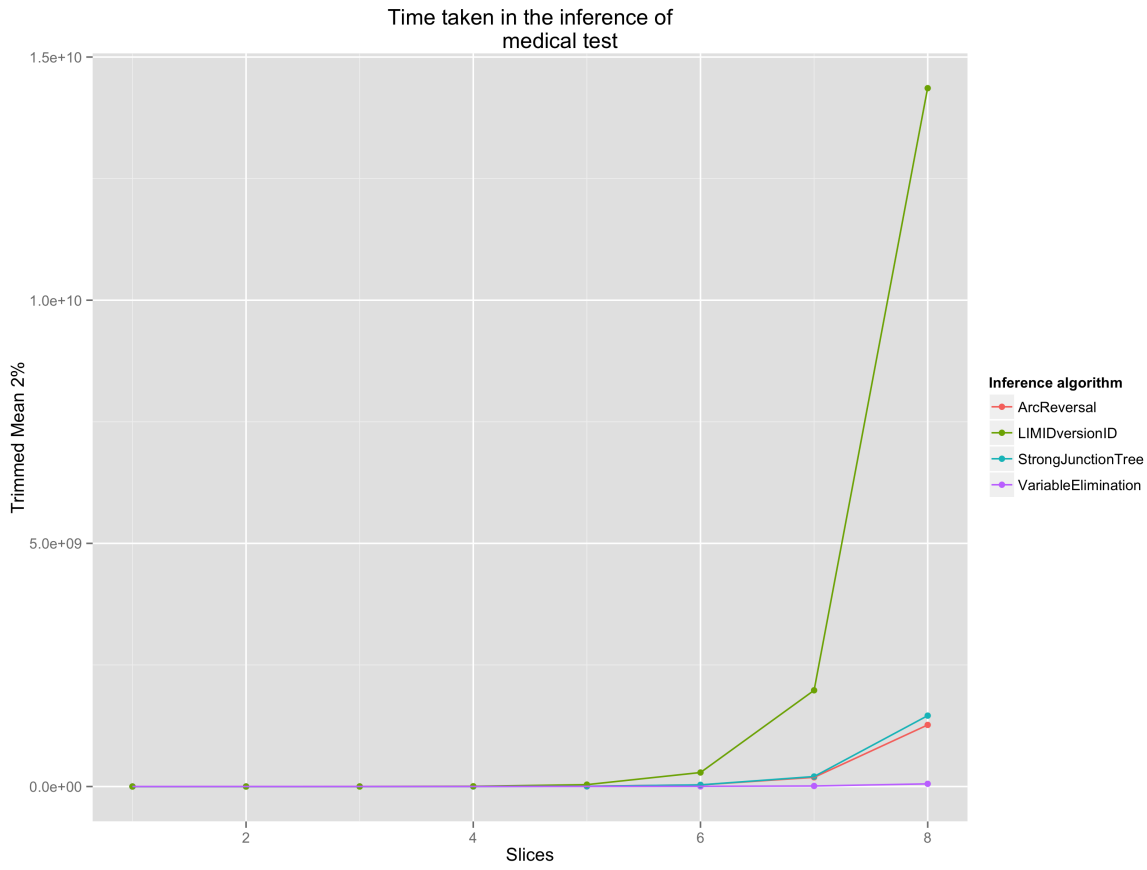

Figure 3.11: Time comparison: trimmed mean 2 vs. slices in the n-test medical decision problem inferences.

operations are complex, thus the poor time efficiency.

The memory performance is no so high in the case of strong junction trees and the n-test medical decision problem because only one clique is involved. And in the case of the follow-up problem the algorithm is beaten by variable elimination and arc reversal in the simpler networks, those till three slices of complexity, because the cliques are not so big, and one of the pre-process of the algorithm is to gather together the potentials assigned to each clique. The multiplication of probability potentials and the sum of utility potential lead to bigger potentials, and this amount of memory is not compensated with the weight of the cliques.

It is necessary to emphasise that in the experiments we have eliminated the messages that were passing along the network, as we only needed to consult the network once. If the comparison had been about the speed with several questions asked to the same network, the messages should have been stored and then the values would vary though it would have been faster than variable elimination and arc reversal.

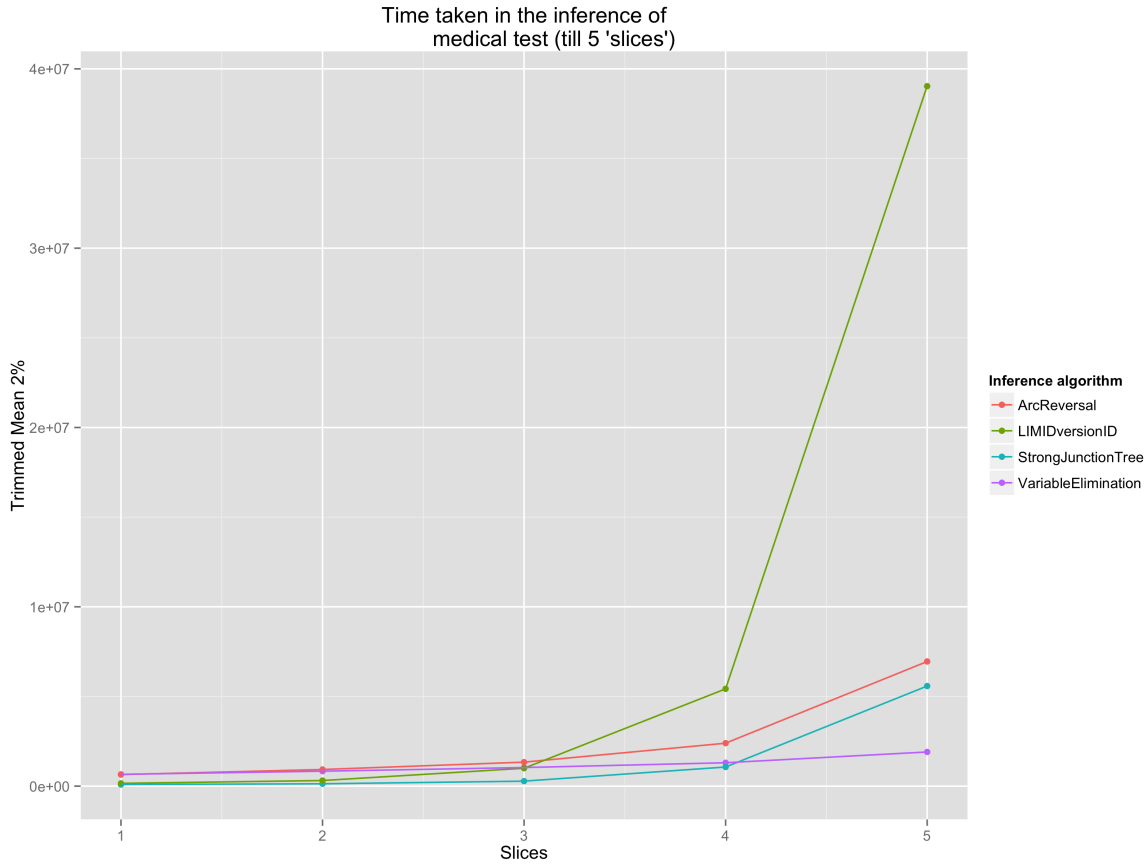

Figure 3.12: Time comparison: trimmed mean 2 vs. slices in the n-test medical decision problem inferences (networks with 5 slices or less).

Finally, the LIMID-conversion algorithm is remarkably the worst of all the algorithms compared. The authors of the algorithm (Nilsson & Lauritzen, 2000; Lauritzen & Nilsson, 2001) stated that their method is better than the strong junction tree as it creates smaller cliques and as a result the complexity of the operations is also smaller. Nevertheless, as we explained in Section 2.4.4 the method requires several iterations, one per decision present in the network. And also it needs to store the messages passed between the cliques in every iteration as they might be needed in subsequent iterations.

The authors, to overcome this difficulty proposed an enhancement to the algorithm so only one flow of the message would be needed. But this strengthening in the algorithm is completely related to the structure of the clique tree and hence to the structure of the network. Actually, in Lauritzen & Nilsson (2001) they show one example that requires going through all the network twice. The results of this algorithm are those that we expected from the beginning and refute the assumptions of its authors.

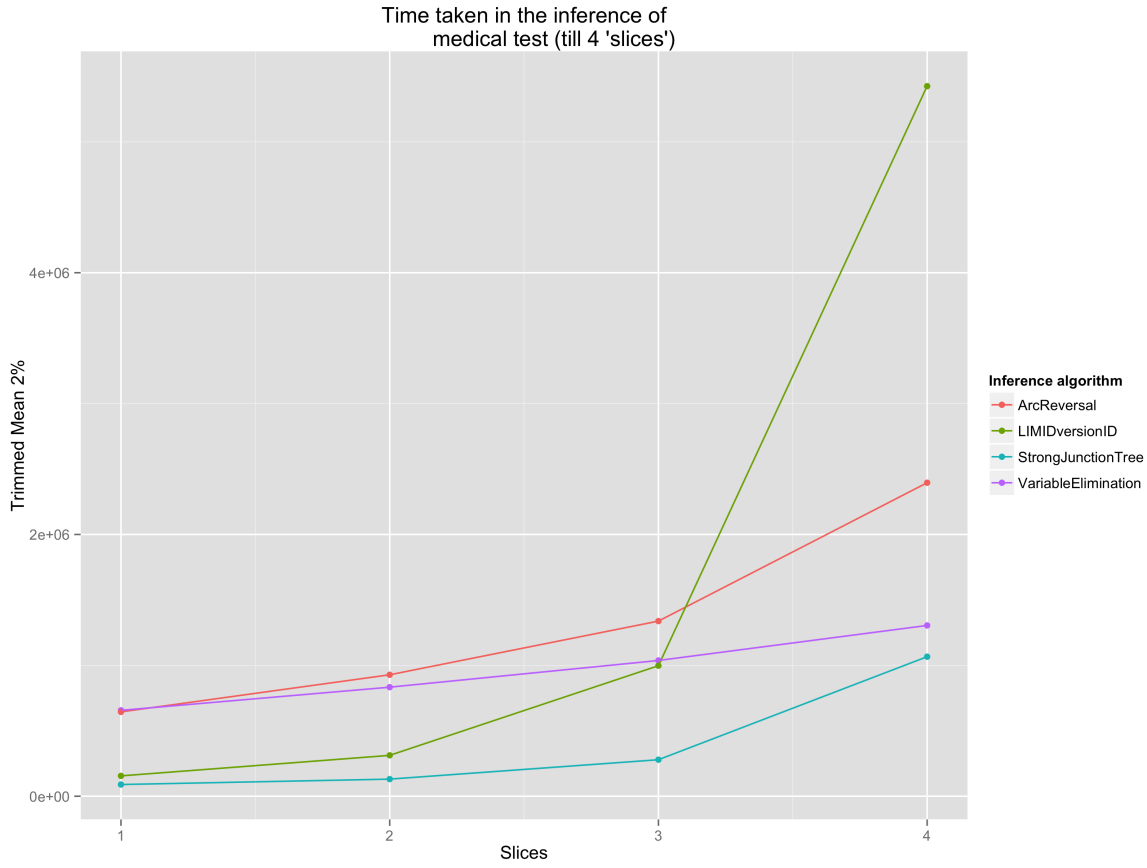

Figure 3.13: Time comparison: trimmed mean 2 vs. slices in the n-test medical decision problem inferences (networks with 4 slices or less).

3.6 Discussion

With the results exposed in Section 3.4, it seems that the LIMID-conversion is not a recommendable algorithm as in the tested networks from the complexity of three slices and over, its performance fell exponentially compared to the other three algorithms.

Putting aside the LIMID-conversion algorithm, and paying attention to the small networks—those of 4 slices or less—, strong junction tree performs slightly better than variable elimination and then it comes arc reversal. We find the same ranking in the memory ratios. As we said in the previous Section, this is likely because the strong junction tree has to calculate the elimination order of the variables just once and can perform several operations in the same clique. For the networks we are speaking about the cliques are small, so the operations inside them are not very complex. In Figures 3.13 and 3.17 the gradient of the speed of the inference of each one of these three algorithms is similar between the networks they are confronted to. It seems that the structure of

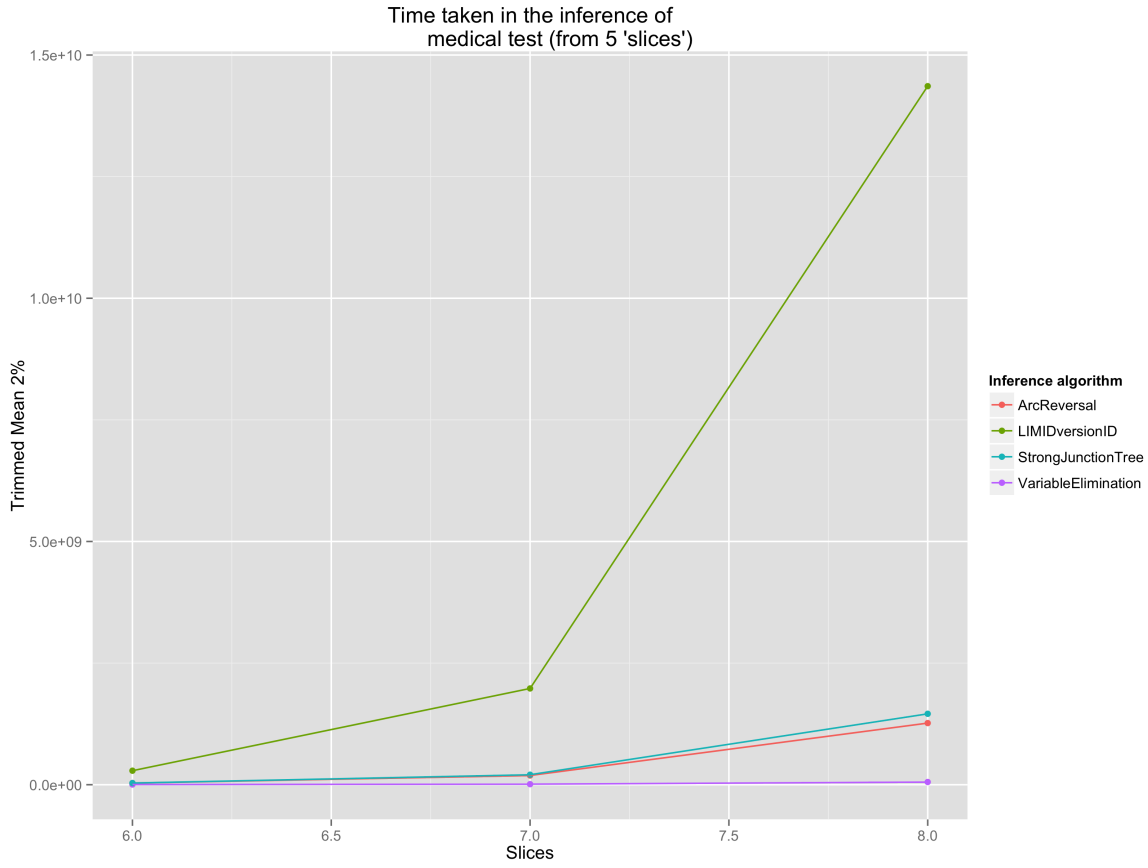

Figure 3.14: Time comparison: trimmed mean 2 vs slices in the n-test medical decision problem inferences (networks with 5 slices or more).

the network is not affecting the speed ratios. But taking in count the average of all the networks (see Table 3.9) the kind of network in use affects both speed and memory consumed.

If we focus now in the biggest networks—those of 5 slices or more—which time comparison graph for the n-test medical decision problem and the follow-up problem are in Figures 3.14 and 3.18 respectively, the scenario changes. Variable elimination becomes the faster algorithm, followed by arc reversal and just only a bit slower than it then it comes strong junction tree. But looking at the memory comparison, the ranking is turned over again and strong junction tree beats the other algorithms. And it outperforms in the case of the follow-up problem. We above framed this numbers in the different structures of the networks, the cliques formed and their separators. This would imply that if the memory is a concern during the evaluation, in *horizontal* networks strong junction tree should be use. It also uses less memory in the *vertical* network but its performances resembles the ones of the other algorithms. As we said before, there are ways to use

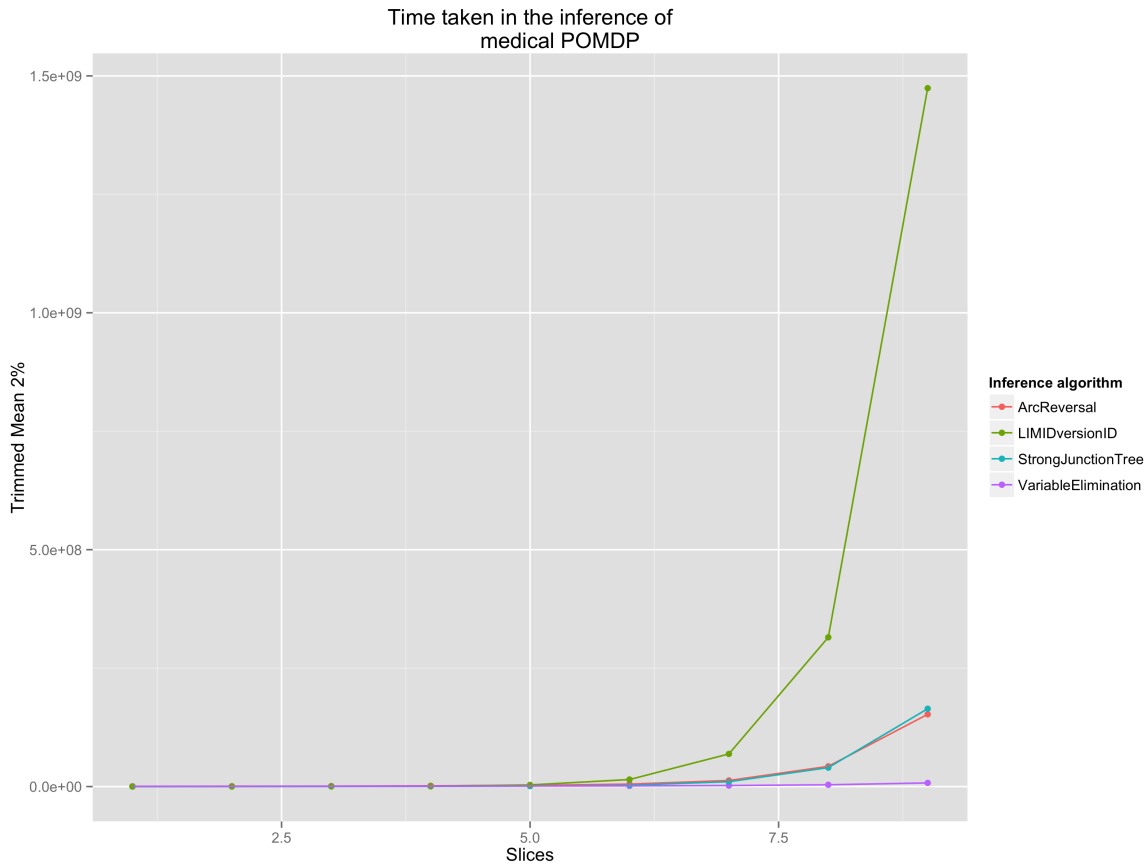

Figure 3.15: Time comparison: trimmed mean 2 vs. slices in the follow-up problem inferences.

the strong junction tree that would require occupying more memory but that also would improve their performance regarding the speed, depending on the scenarios. In any case, these comparisons are not within the scope of this dissertation.

From all the above, it could be inferred that variable elimination algorithm is the most balanced algorithm for both kinds of networks and for their different sizes when addressing just one question to the network: maximum expected utility and optimal policy.

3.7 Results reproducibility

As we stated in the introduction of this thesis, we wanted the research community to be able to check the data we obtained and, if desired, reproduce the research and even to contribute to it. In the repository `https://bitbucket.org/artasom/master-thesis-reproducible-research/` anyone will find:

1. The complete database used to store the experiments with the data obtained from

Table 3.7: Cliques of the strong junction tree algorithm in the different networks.

Slices	n-test medical decision problem	Follow-up problem
1	1 clique of 4 variables	2 cliques of 3 variables each
2	1 clique of 6 variables	3 cliques of 5, 4 and 3 nodes
3	1 clique of 8 variables	4 cliques of 7, 5, 4 and 3 nodes
4	1 clique of 10 variables	5 cliques of 9, 6, 5, 4 and 3 nodes
5	1 clique of 12 variables	6 cliques of 11, 7, 6, 5, 4 and 3 nodes
6	1 clique of 14 variables	7 cliques of 13, 9, 6 (2 of them), 5, 4 and 3 nodes
7	1 clique of 16 variables	8 cliques of 15, 9, 8, 6 (2 of them), 5, 4 and 3 nodes
8	1 clique of 18 variables	9 cliques of 17, 10, 9, 6 (3 of them), 5, 4 and 3 nodes
9	-	10 cliques of 19, 12, 9, 8, 6 (3 of them), 5, 4 and 3 nodes

Table 3.8: Cliques of the LIMID-conversion algorithm in the different networks.

Slices	n-test medical decision problem	Follow-up problem
1	1 clique of 4 variables	2 cliques of 3 variables each
2	1 clique of 6 variables	3 cliques of 5, 4 and 3 nodes
3	1 clique of 8 variables	4 cliques of 7, 5, 4 and 3 nodes
4	1 clique of 10 variables	5 cliques of 9, 6, 5, 4 and 3 nodes
5	1 clique of 12 variables	6 cliques of 11, 7, 6, 5, 4 and 3 nodes
6	1 clique of 14 variables	7 cliques of 13, 9, 6 (2 of them), 5, 4 and 3 nodes
7	1 clique of 16 variables	8 cliques of 15, 10, 7, 6 (2 of them), 5, 4 and 3 nodes
8	1 clique of 18 variables	9 cliques of 17, 10, 9, 6 (3 of them), 5, 4 and 3 nodes
9	-	10 cliques of 19, 11, 10, 7, 6 (3 of them), 5, 4 and 3 nodes

them. The connection to the database can be done with the user `artasom` and the password `artasom`.

2. Also, the data in `csv` format.

3. The described programs `ACGenerateNetworks` and `ACGenerateStatistics` in executable `jar` format.

Table 3.9: Computational time and memory used ratio comparison of the n-test medical decision problem over the follow-up Problem (till slice number 8).

Algorithm	Time comparison	Memory comparison
Variable elimination	3,795376545	18,09639944
Arc reversal	7,453063889	17,60600901
Strong junction tree	9,906587178	148,7520652
LIMID-conversion	14,49292242	16,44713659

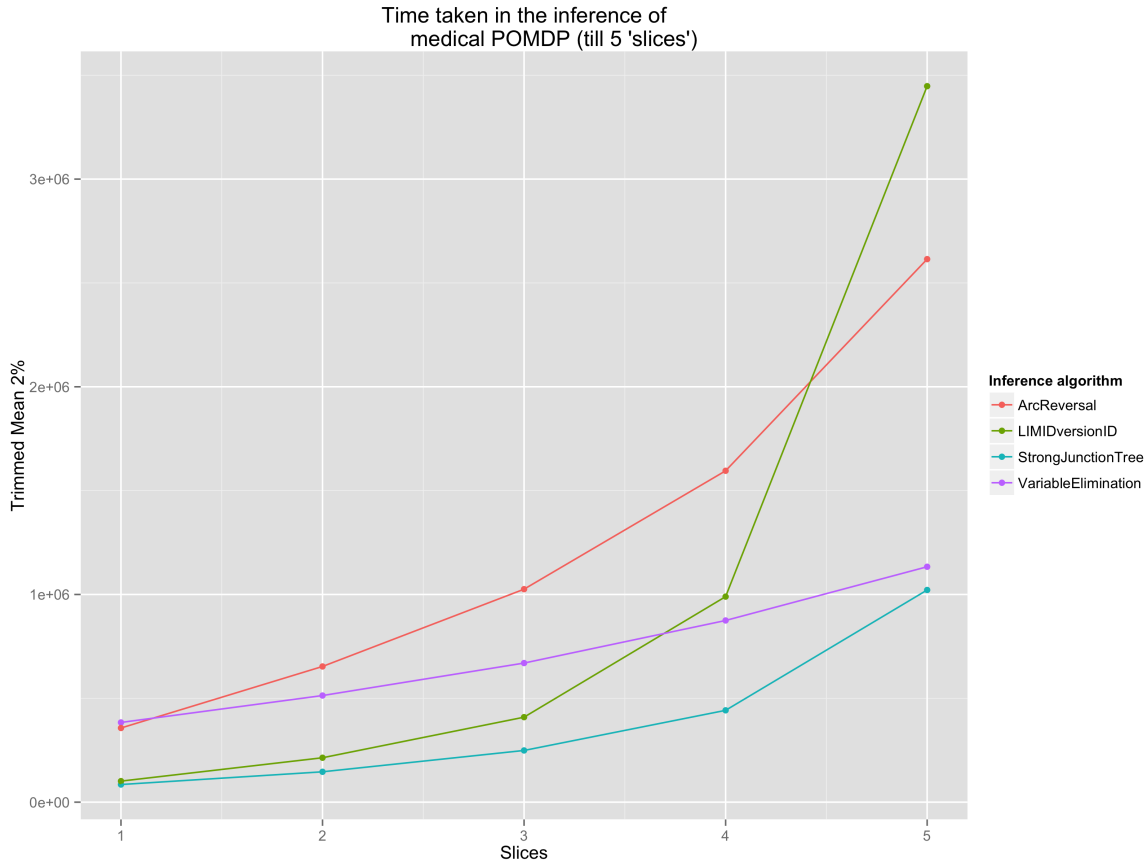

Figure 3.16: Time comparison: trimmed mean 2 vs. slices in the follow-up problem inferences (networks with 5 slices or less).

Table 3.10: Memory comparison of the n-test medical decision problem.

Slices	Memory AR / VE	Memory AR / SJT	Memory AR / LIMID
1	1	1	0,625
2	1,053571429	1,340909091	0,7375
3	1,064220183	1,5	0,776785714
4	1,066221766	1,564006024	0,791539634
5	1,066586683	1,5875	0,797003074
6	1,066652379	1,595737859	0,798961901
7	1,066664126	1,598560987	0,799645795
8	1,066666217	1,599516823	0,799880288

4. The OpenMarkov networks used to compare the efficiency of the algorithms.

5. The outcome of the R programming language: the excel-sheets in `xls` and `csv` formats and the plots in `png` format.

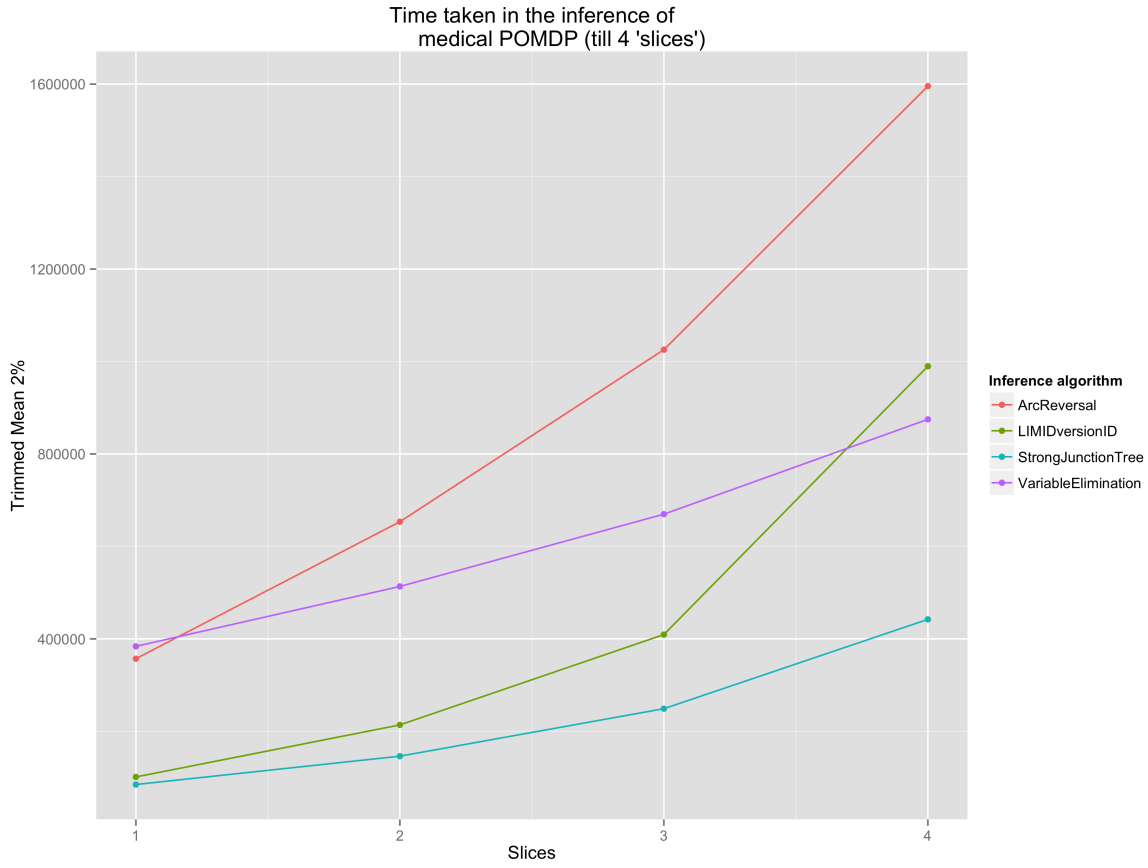

Figure 3.17: Time comparison: trimmed mean 2 vs. slices in the follow-up problem inferences (networks with 4 slices or less).

6. The RMySQL library used to read the data from the database into R.

7. The R project of RStudio[5] and an RMarkdown file to compile with knitr (Xie, 2013) the outcome of the R scripts used in HTML.

8. The R scripts generated to read the data from the database, structure it, generate the numerical comparisons, store the excel-sheet files and the plots.

9. The unix scripts used to launch the java programs, where anyone can consult also the parameters requested by the programs.

With all theses resources, and once the database is up & running in the machine, anyone can replicate the results we obtained. In order to access the source code of OpenMarkov, any researcher may head to `http://www.openmarkov.org/` and request for access to the repository where a full copy of OpenMarkov is located.

[5] `http://www.rstudio.com/products/RStudio/`

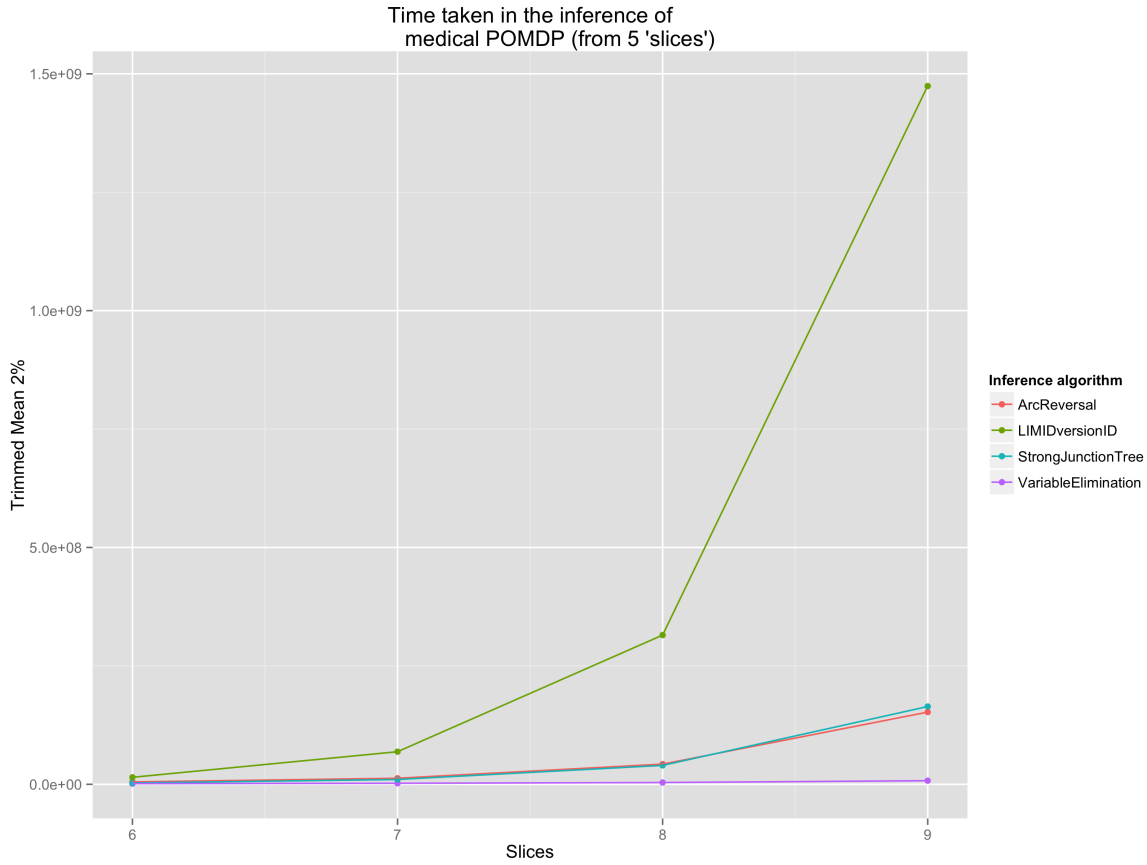

Figure 3.18: Time comparison: trimmed mean 2 vs. slices in the follow-up problem inferences (networks with 5 slices or more).

Table 3.11: Memory comparison of the follow-up problem.

Slices	Memory AR / VE	Memory AR / SJT	Memory AR / LIMIDs
1	0,555555556	0,416666667	0,5
2	0,764705882	0,433333333	0,295454545
3	1,1875	1,096153846	0,483050847
4	1,109375	2,476744186	0,605113636
5	1,110677083	3,985981308	0,71440536
6	1,112304688	9,285326087	0,755639098
7	1,111083984	11,91361257	0,79655776
8	1,111104329	24,68942134	0,820655767
9	1,111109416	26,04351454	0,827479754

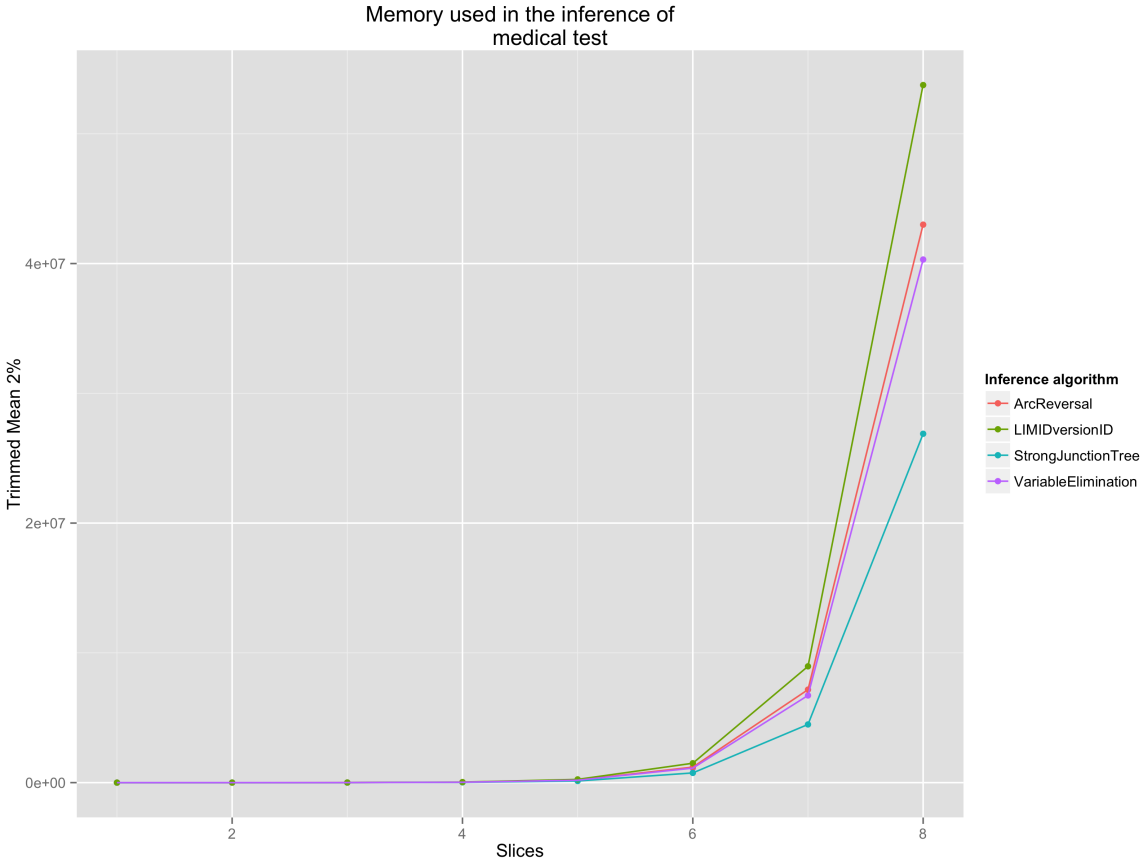

Figure 3.19: Memory comparison of the n-test medical decision problem inferences.

Table 3.12: Machine hardware information.

Model Name	MacBook
Model Identifier	MacBook6,1
Processor Name:	Intel Core 2 Duo
Processor Speed	2,26 GHz
Number of Processors	1
Total Number of Cores	2
Memory	8 GB
OS X Version	10.9.4

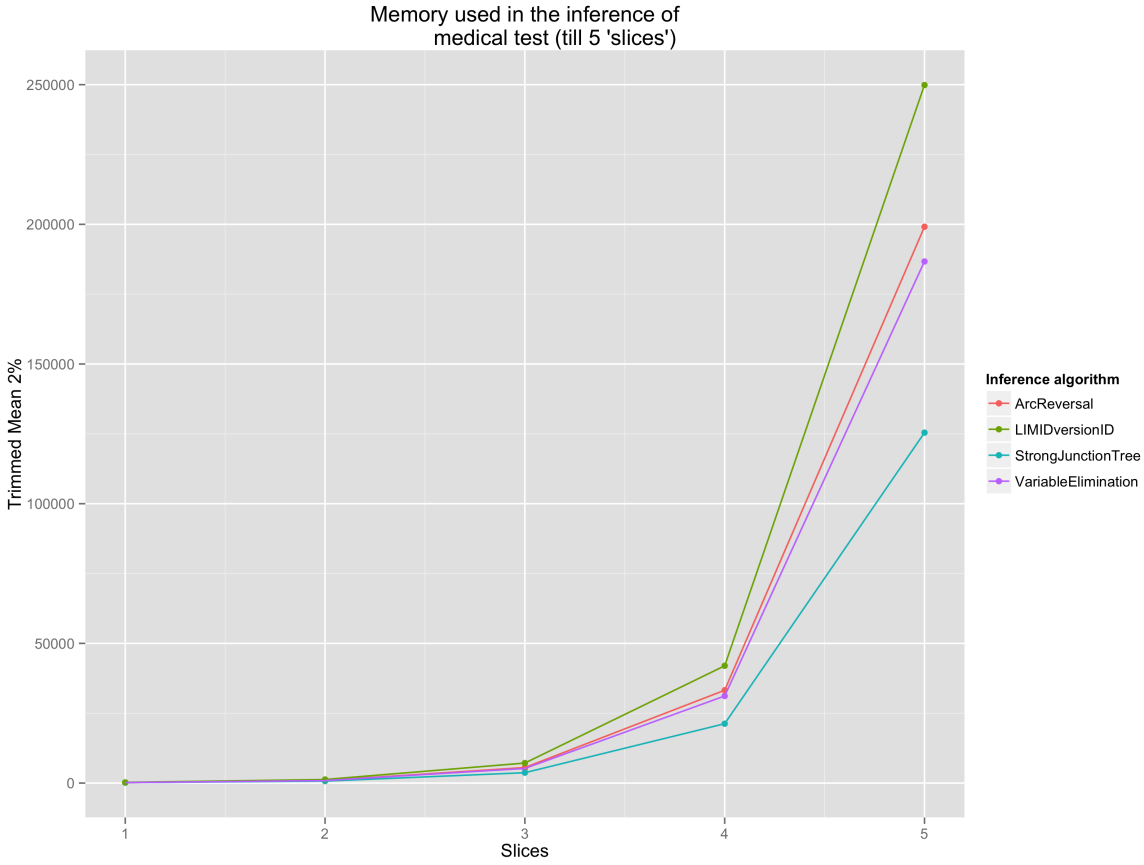

Figure 3.20: Memory comparison of the n-test medical decision problem inferences (networks with 5 slices or less).

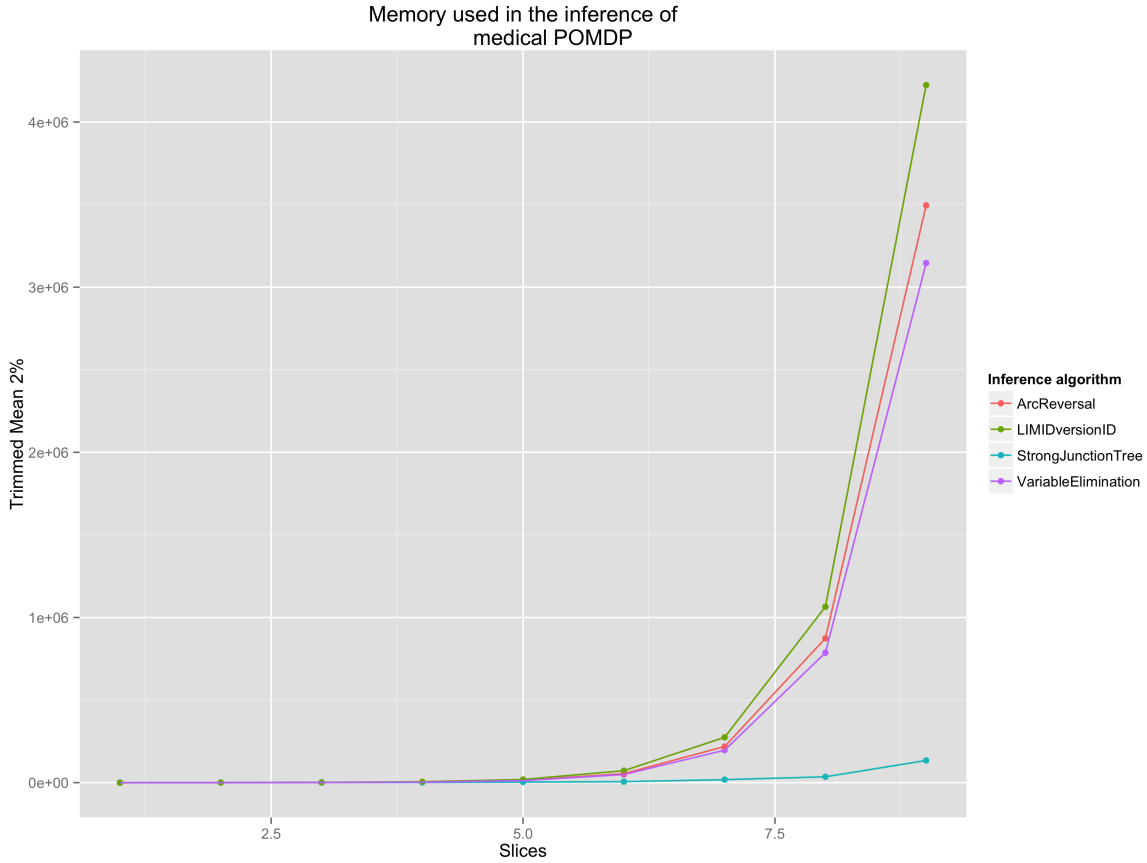

Figure 3.21: Memory comparison of the follow-up problem inferences.

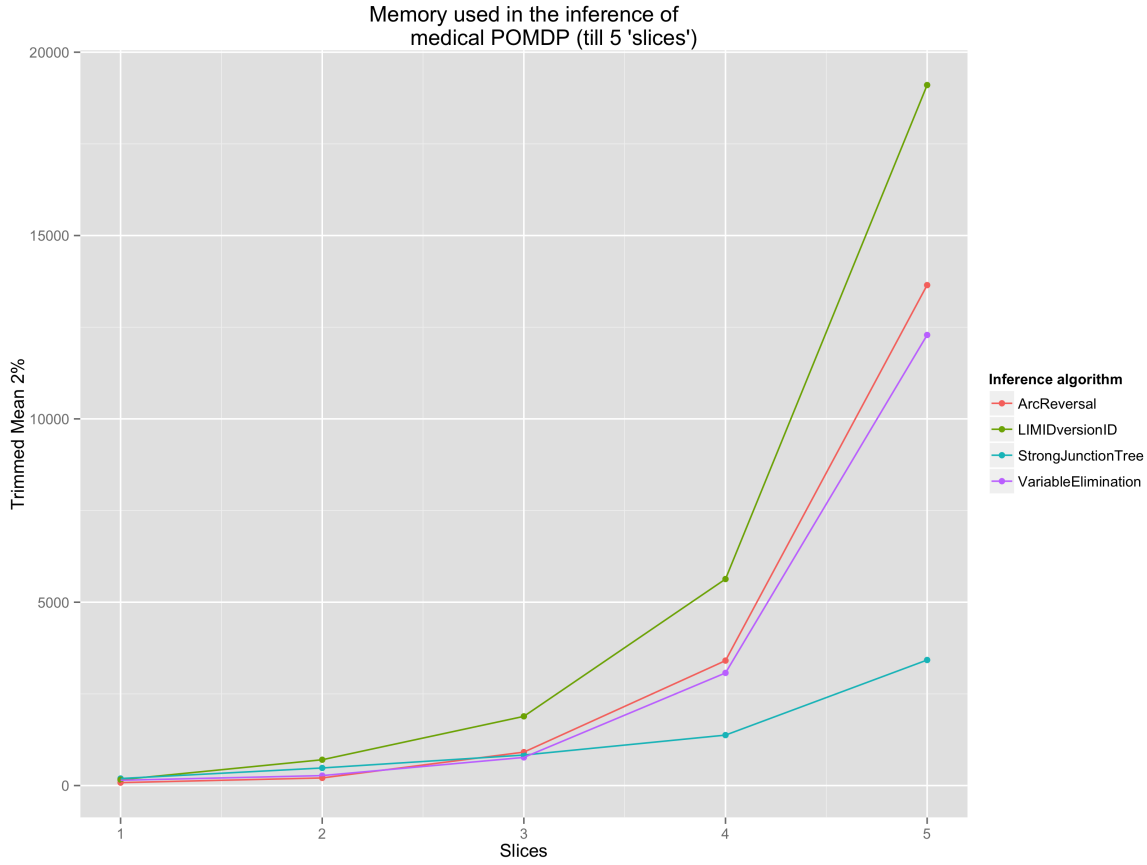

Figure 3.22: Memory comparison of the follow-up problem inferences (networks with 5 slices or less).

Chapter 4

Conclusions and future work

4.1 Conclusions

The results we obtained of the performance of variable elimination and arc reversal are analogous to those given by other authors (Luque & Díez, 2010; Butz et al., 2009b). Our experiment of the comparison with the other two algorithms is novel and provides the first empirical evidence of the performance of the LIMID-conversion algorithm.

As we have seen in Chapter 3, no algorithm of the studied is a winner in all the scenarios. Despite this fact, some recommendations emerge from the inferences performed. The graphics and tables pulled out of the comparison reveal that with small networks there is little difference between the algorithms. But with bigger networks, if memory is an issue then the strong junction tree algorithm ought to be used; it will give outstanding results with *horizontal* networks. If a balance between memory and speed is sought, variable elimination should be used as it is remarkably faster than the others in all the big networks analysed. Generally speaking, the LIMID-conversion of an ID must be avoided because, contrary to the claim of Nilsson & Lauritzen (2000) it does not "yield significant savings of memory and computational time when compared to traditional methods".

Also of note is the comparison of the time employed between the two types of networks tested. The ratios show that all the algorithms perform the inference faster in the follow-up problem than in the n-test medical decision problem. In the case of the algorithms that use junction tree structures, the inference is at least 10 times faster. Regarding the memory, the n-test medical decision problem requires 18 times more memory than the follow-up problem, except in the case of the strong junction tree where the memory used by this network compared to the other one skyrockets to 148 times.

In the scope of this thesis, we have implemented in Java and put at the disposal

of the Artificial Intelligence community three classic algorithms to perform inference in influence diagrams. With the aim of giving more power to this research, we have pursued its reproducibility. A public repository has been set where any expert, researcher, or student can find all the data used in the experiments as well as the files necessary to reproduce the research. We think this could be very useful for people involved the field of PGMs.

4.2 Future work

The main lines for the future work related to this dissertation are to include some improvements on the algorithms and compare them again. For example, the possibility of avoiding redundant variables could be introduced in the strong junction tree algorithm. There is also some ongoing work regarding the election of the elimination order of the variables, an NP-complete problem, that may be implemented in variable elimination. Also, there is some research on lazy evaluation for IDs (Madsen & Jensen, 1999; Vomlelová & Jensen, 2004), which can be combined within variable elimination and arc reversal. There are contradictory recent reports about arc reversal performing better than variable elimination under certain scenarios when using lazy propagation (Butz et al., 2011).

It would be also also interesting to code more algorithms into OpenMarkov, such as Branch-and-Bound Search (Yuan et al., 2010) and Decision Circuits (Shachter & Bhattacharjya, 2010) and incorporate them to the comparison. Basically, as we have developed a framework to compare and contrast different algorithms, any improvements on the existing ones or any implementation of new algorithms, can be included in new analyses.

Bibliography

Arias, M. & Díez, F. J. (2008). Carmen: An open source project for probabilistic graphical models. In F. V. Jensen & U. Kjærulff (Eds.), *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models (PGM'08)* (pp. 25–32). Hirtshals, Denmark.

Arias, M., Díez, F. J., & Palacios, M. P. (2011). *ProbModelXML. A format for encoding probabilistic graphical models.* Technical Report CISIAD-11-02, UNED, Madrid, Spain.

Arias, M., Díez, F. J., Palacios-Alonso, M. A., & Bermejo, I. (2012). ProbModelXML. a format for encoding probabilistic graphical models. In Cano et al. (2012), (pp. 11–18).

Barber, D. (2012). *Bayesian Reasoning and Machine Learning.* Cambridge University Press.

Bermejo, I., Oliva, J., Díez, F. J., & Arias, M. (2012). Interactive learning of Bayesian networks with OpenMarkov. In Cano et al. (2012), (pp. 27–34).

Butz, C., Konkel, K., & Lingras, P. (2009a). Join tree propagation utilizing both arc reversal and variable elimination. In *Florida Artificial Intelligence Research Society Conference.*

Butz, C. J., Chen, J., Konkel, K., & Lingras, P. (2009b). A formal comparison of variable elimination and arc reversal in bayesian network inference. *Int. Dec. Tech.*, 3(3), 173–180.

Butz, C. J., Konkel, K., & Lingras, P. (2011). Join tree propagation utilizing both arc reversal and variable elimination. *International Journal of Approximate Reasoning*, 52, 948–959.

Cano, A., Gómez, M., & Nielsen, T. D., Eds. (2012). *Proceedings of the Sixth European Workshop on Probabilistic Graphical Models (PGM'12)*, Granada, Spain.

Cooper, G. F. (1988). A method for using belief networks as influence diagrams. In R. D. Shachter, T. Levitt, L. N. Kanal, & J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 4 (UAI'88)* (pp. 55–63). Amsterdam, The Netherlands: Elsevier Science Publishers.

Grunwald, P. & Spirtes, P., Eds. (2010). *Proceedings of the Twenty-sixth Conference on Uncertainty in Artificial Intelligence (UAI'10)*, Corvallis, OR. AUAI Press.

Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In G. Cooper & S. Moral (Eds.), *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI'98)* (pp. 211–219). San Francisco, CA: Morgan Kauffmann.

Howard, R. A. & Matheson, J. E. (1984). Influence diagrams. In R. A. Howard & J. E. Matheson (Eds.), *Readings on the Principles and Applications of Decision Analysis* (pp. 719–762). Menlo Park, CA: Strategic Decisions Group.

Huang, C. & Darwiche, A. (1996). Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15, 225–263.

Jensen, F., Jensen, F. V., & Dittmer, S. L. (1994). From influence diagrams to junction trees. In R. L. de Mantaras & D. Poole (Eds.), *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI'94)* (pp. 367–373). San Francisco, CA: Morgan Kauffmann.

Jensen, F. V. & Nielsen, T. D. (2007). *Bayesian Networks and Decision Graphs*. New York: Springer-Verlag, second edition.

Kjærulff, U. (1993). *Aspects of Efficiency Improvement in Bayesian Networks*. PhD thesis, Dept. Mathematics and Computer Science, Aalborg University, DINA Research Report No. 39.

Kjærulff, U. & Madsen, A. L. (2010). *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. New York: Springer.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

Koller, D. & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: The MIT Press.

Lauritzen, S. L. & Nilsson, D. (1999). Limids of decision problems.

Lauritzen, S. L. & Nilsson, D. (2001). Representing and solving decision problems with limited information. *Management Science*, 47, 1235–1251.

Lauritzen, S. L. & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50, 157–224.

Luque, M. & Díez, F. J. (2010). Variable elimination for influence diagrams with super-value nodes. *International Journal of Approximate Reasoning*, 51, 615 – 631.

Madsen, A. & Jensen, F. V. (1999). Lazy evaluation of symmetric Bayesian decision problems. In K. Laskey & H. Prade (Eds.), *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)* (pp. 382–390). San Francisco, CA: Morgan Kauffmann.

Madsen, A. L. & Nilsson, D. (2001). Solving influence diagrams using HUGIN, Shafer-Shenoy and lazy propagation. In J. Breese & D. Koller (Eds.), *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01)* (pp. 337–345). San Francisco, CA: Morgan Kauffmann.

Nilsson, D. & Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In C. Boutilier & M. Goldszmidt (Eds.), *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)* (pp. 436–445). San Francisco, CA: Morgan Kauffmann.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Shachter, R. D. (1986). Evaluating influence diagrams. *Operations Research*, 34, 871–882.

Shachter, R. D. (1988). Probabilistic inference and influence diagrams. *Operations Research*, 36, 589–605.

Shachter, R. D. & Bhattacharjya, D. (2010). Dynamic programming in influence diagrams with decision circuits. In Grunwald & Spirtes (2010), (pp. 509–516).

Stigler, S. M. (1973). The asymptotic distribution of the trimmed mean. *The Annals of Statistics*, 1(3), pp. 472–477.

Tatman, J. A. & Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20, 365–379.

Tversky, A. & Kahneman, D. (1974). Judgement under uncertainty: Heuristics and biases. *Science*, 185, 1124–1131.

Vomlelová, M. & Jensen, F. V. (2004). An extension of lazy evaluation for influence diagrams avoiding redundant variables in the potentials. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 12, 1–17.

Widenius, M. & Axmark, D. (2002). *Mysql Reference Manual*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1st edition.

Xie, Y. (2013). *Dynamic Documents with R and knitr*. Boca Raton, Florida: Chapman and Hall/CRC. ISBN 978-1482203530.

Yuan, C., Wu, X., & Hansen, E. (2010). Solving multistage influence diagrams using branch-and-bound search. In Grunwald & Spirtes (2010), (pp. 691–700).