UNIVERSIDAD DE CASTILLA - LA MANCHA
Departamento de Sistemas Informáticos

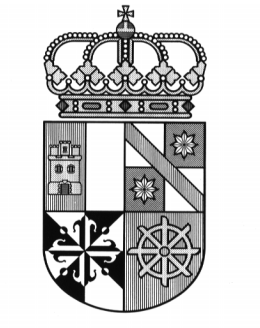
Doctoral Thesis

Bayesian networks inference: Advanced algorithms for triangulation and partial abduction

M. Julia Flores Gallego

Albacete, November 2005

UNIVERSIDAD DE CASTILLA - LA MANCHA



Bayesian networks inference: Advanced algorithms for triangulation and partial abduction

PhD Thesis dissertation

Departamento de Sistemas Informáticos

University of Castilla - La Mancha (UCLM)

European Doctorate

María Julia Flores Gallego

Supervised by

Dr. José Antonio Gámez Martín (UCLM)

Dr. Serafín Moral Callejón (Universidad de Granada)

Albacete, November 2005

Acknowledgments

The development of this work can be seen as a long journey through study, knowledge, discovery, endeavour,... that I definitively could never have done alone. There are many people who, to a greater or lesser extent, have contributed to the accomplishment of this thesis, my most important professional goal so far.

I will probably not be able to list every person, but I would at least like to express my gratitude explicitly to those who have been outstanding colleagues either along this whole *journey* or in certain key, and sometimes hard, moments.

First of all, I can not start without thanking my supervisors José Antonio Gámez Martín and Serafín Moral Callejón, who certainly have given me the best guidance and advice I could ever have received. I must say I appreciate all the time they have devoted to proposing and discussing ideas, their bid to improve the designed methods and thoroughly read and review these pages (even on their holidays). José A. is a wonderful person to work with, he has treated me as an equal, being at the same time a great *boss* who directed my work along the right path when necessary. On the other hand, Serafín, in spite of the physical distance between Albacete and Granada, has shown his proximity and involvement in the fulfilment of this project with visits, meetings, and constant communication with us. Apart from their valuable support, help and effort, I should also mention my admiration for their human touch.

This journey has mainly taken place in the "Departamento de Sistemas Informáticos" in Albacete, which has been a pleasant environment to develop my work. My research group SIMD has been an important forum to discuss and learn, and I wish to thank all of its members for their support, especially at the final stages of this thesis. My department colleagues have also made the day-to-day problems and routine easier. I would like to mention those who have especially influenced me with their professional help, but even further with their friendship: Molina (my colleague since we started university), Kike and Blanca (who gave us the warmest welcome to the department and still take great care over social relations in the workplace), Encarni, Isma, Luis and the other fellow *PhD travellers* Víctor, Paco and Goyito with who I have shared both sweet and bitter moments. I want to specially thank Elena for her loyalty always finding time to help me either with technical difficulties or personal ones. I would like to extent my gratitude to all the *mazmorreros* as well as to the rest of my colleagues, from the highest to the lowest.

My deepest gratitude to Finn V. Jensen and all members of the research group Decision Support Systems Unit in the computer science department of Aalborg University. Every time I have visited them, I have been welcomed with open arms, providing me with the best working conditions. I found my stay in the Spring semester 2003 particularly profitable, where I could learn from the numerous group's workshops and seminars. I also wish to thank

their approachable attitude to solving my questions or doubts whenever I got stuck. Thanks to Olav Bangsø because he gave me an insight into a completely new theory approach. Kristian G. Olesen deserves a very special note of thanks for his assistance, advice and support, but also his great kindness and geniality. He is always willing to help in spite of his usual tight schedule. Our e-mail collaboration during the last five years has also provided valuable feedback for me. I can not remember Denmark without mentioning Jens D. Nielsen and Jose Peña, who made my stay in Aalborg even nicer, I recall having many interesting and funny conversations as well as a few beers.

I must mention my short visit to Granada University, where again I was received in a congenial atmosphere. In the course of these weeks, I could enjoy a helpful learning environment. Special thanks, apart from Serafín, to my office partner Juan Huete, and also to Andrés Cano, Manolo Gómez and Luis M. De Campos. These last three have also been collaborators in the research project "Elvira II" which has given me not only quite useful financial support but also the opportunity to attend such productive meetings and, furthermore, meeting very good researchers and nice people from different Spanish universities.

During these *travel* years, my personal life has also been of great relevance to the achievement of this work. My family has always offered me their whole-hearted support and they have encouraged me to persevere on the path to the completion of this thesis. They have suffered my changing moods derived from the status of the thesis on most occasions. Work and travel have deprived me of spending more time with them, but they have always been very understanding and supportive. Thanks, mum, for being so proud of me, this confidence always fed my enthusiasm for this project. Two very special people who deserve mention are my sister, Marilén, and my little nephew Pablo. The former has given me the education, also entertainment, when I was younger and still today she regales me with great *fight* & fun moments. The latter came in the middle and busiest period of this thesis. Even though I can not spend as much time with him as I would like to, he is always ready to offer me his best smile, enough to cheer me up more than any of the words that he cannot pronounce yet.

Finally, I could devote several pages to those good friends who have accompanied me during these years. I am afraid that would be too much, sorry if I haven't named you, it does not matter, because you already know how important you have been for me: when someone feels good it is much easier to work better. I will make an effort to summarise people in groups: of course thanks to las bolas (and los bolos as well) for being always so close, the enterprising French course team, the traveller Murcianicas, that incredible group of Spanish Power together in spite of distance, Caillebotte family, my Little Darlings (do not think badly), las FAVORIRAS... I need to mention two particular friends who have always been there, from the very beginning until today, every time I needed them: *Glo* (thanks for your attentiveness) and Belén (thanks for all your listening and understanding).

Agradecimientos

El desarrollo de este trabajo podría verse como un largo trayecto a través del estudio, conocimiento, descubrimiento, esfuerzo, ... que sin duda jamás podría haber recorrido sola. Hay mucha gente que, en mayor o menor medida, ha hecho posible la realización de esta tesis, el objetivo profesional más importante que me he marcado hasta hoy. Probablemente no podré enumerar a todas estas personas, pero al menos me gustaría expresar mi agradecimiento explícitamente a aquellos cuya compañía a lo largo de este *viaje* ha sido especialmente destacada o que han estado a mi lado en momentos clave, y a veces difíciles.

No podría empezar sino dando las gracias a mis directores José Antonio Gámez Martín y Serafín Moral Callejón. Los dos me han guiado con gran maestría y me han dado sabios consejos. Debo decir que valoro especialmente todo el tiempo que han dedicado a proponer y discutir ideas, su buena disposición para mejorar los métodos diseñados y para leer y revisar concienzudamente estas páginas (incluso durante sus vacaciones). Trabajar con José A. es una suerte, siempre me ha tratado como una igual, consiguiendo al mismo tiempo ser un *jefe* estupendo que ha llevado siempre mi trabajo por el buen camino y lo ha reconducido cuando ha sido necesario. Por otro lado, Serafín, a pesar de la distancia entre Albacete y Granada, ha demostrado siempre su cercanía y su implicación en este proyecto mediante visitas, reuniones y una comunicación constante con nosotros. Además de su valiosa ayuda, apoyo y esfuerzo, debería mencionar también mi admiración por la gran calidad humana de los dos.

Este *recorrido* ha transcurrido principalmente en el "Departamento de Sistemas Informáticos" en Albacete, ciertamente un agradable entorno donde desarrollar mi trabajo. Mi grupo de investigación (SIMD) ha supuesto un foro importante donde discutir ideas y aprender, y me gustaría agradecer a todos sus integrantes el apoyo que de ellos he recibido, especialmente al final. Igualmente mis compañeros de departamento han conseguido que los problemas del día a día y la rutina resultaran más fáciles de llevar. Quisiera mencionar a aquellos que han tenido una influencia especial por su ayuda profesional, y aún más lejos por su trato personal: Molina (compañero desde el inicio de la carrera), Kike y Blanca (nos acogieron muy cariñosamente desde el principio y aún hoy siguen cuidando con esmero las relaciones sociales entre los compañeros), Encarni, Isma, Luis y mis *colegas* doctorandos Víctor, Paco y Goyito con quienes he compartido momentos buenos y malos. Quiero agradecer de manera especial a Elena su amistad y fidelidad encontrando siempre tiempo para echarme un cable ya fuese con problemas técnicos o personales. Me gustaría además extender mi gratitud a todos los *mazmorreros* así como al resto de mis compañeros, desde el primero al último.

Mi más profundo agradecimiento a Finn V. Jensen y a todos los miembros del grupo de investigación de *Sistemas de Ayuda a la Decisión* (DSS) del departamento de informática en la universidad de Aalborg. Siempre que les he visitado, me han recibido con gran hospitalidad, y me han facilitado unas condiciones de trabajo inmejorables. Mi estancia en el 2003 fue en

especial fructífera, ya que tuve la oportunidad de aprender de sus numerosos seminarios y reuniones. Asimismo quisiera agradecerles su accesibilidad para la resolución de mis dudas o preguntas en los momentos en los que no sabía por dónde o cómo continuar. Gracias a Olav Bangsø que me ayudó a conocer y entender un enfoque teórico completamente nuevo para mí. Por supuesto, Kristian G. Olesen merece especial mención y mi sincero reconocimiento por su ayuda, consejo y ánimo, pero también por su increíble amabilidad y simpatía. Él siempre está dispuesto a contribuir con su trabajo, aun a pesar de su tan apretada agenda. Nuestra colaboración de los últimos cinco años (principalmente vía e-mail) ha sido muy enriquecedora. No podría recordar Aalborg sin nombrar a Jens D. Nielsen y Jose Peña. Ellos hicieron que mi estancia en Dinamarca fuese aún más agradable, tengo en mi memoria muchas conversaciones interesantes y divertidas con ellos, así como alguna que otra cerveza.

Mi visita a Granada también es digna de mención. De nuevo allí fui recibida en un ambiente cordial y agradable. Quiero destacar, además de a Serafín, a mi compañero de oficina Juan Huete y en particular a Andrés Cano, Manolo Gómez y Luis M. De Campos. Los tres últimos han sido también colaboradores en el proyecto nacional de investigación "Elvira II". Gracias a este proyecto he recibido ayuda económica para mi investigación y he podido asistir a sus tan provechosas reuniones, posibilitándome además el conocer y trabajar con grandes investigadores y muy buena gente de varias universidades españolas.

Durante estos años de *camino*, mi vida personal ha sido determinante para la consecución de este trabajo. Mi familia me ha apoyado siempre incondicionalmente y ellos me han animado a seguir en la senda de la finalización de mi tesis. Son los que han tenido que sufrir mis constantes cambios de humor, muchas veces derivados del estado de la tesis. Mi trabajo y mis ausencias me han privado de pasar más tiempo con ellos, sin embargo siempre se han mostrado comprensivos y me han ofrecido su ayuda. Gracias, mamá, por estar siempre tan orgullosa de mí, esa seguridad ha fomentado mi entusiasmo por este proyecto. Quisiera nombrar a mi hermana Marilén y mi sobrinete Pablo. La primera se ocupó mucho de mí cuando era más joven y aún hoy me concede grandes momentos de diversión y *peleas*. El segundo llegó a mitad de esta tesis, y en el periodo más atareado. Si bien no puedo pasar con él tanto tiempo como yo quisiera, el siempre está dispuesto a ofrecerme su mejor sonrisa, capaz de transmitirme más energía que cualquiera de las palabras que aún no es capaz de pronunciar.

Podría dedicar varias páginas a todos los amigos que me han acompañado en estos años. Como sería excesivo, perdón si no te nombro explícitamente, tú ya sabes el papel que has desempeñado para mí: cuando alguien se siente bien es mucho más fácil trabajar mejor. Me esforzaré en resumir y agruparé a estas personas: gracias a las bolas (y bolos) por estar ahí, a las estudiantes de francés, a las Murcianicas viajeras, al genial grupo de los Spanish Power (juntos aunque lejos), a los Caillebotte, a mis cariñines (que nadie piense mal), a las FAVORIRAS... Tengo que mencionar a dos personas en particular, cuya amistad y apoyo me han acompañado en todo momento, desde el primer día hasta hoy, siempre que lo he necesitado: *Glo* (gracias por estar siempre tan atenta) y Belén (gracias por escucharme tanto y por tu comprensión).

To my parents
(*A mis padres*)

Contents

Introduction

During the last decades, a remarkable development has taken place in the field of artificial intelligence and, particularly, for expert systems. These systems attempt to model concrete problems of the real world so that they face up to a certain restricted situation in a way similar to a human's behaviour.

In order to model a specific problem it is indispensable to consider both uncertainty and vagueness, which are inherent to nature itself. The treatment of uncertainty is one of the necessary tools for an expert system. Probabilities are among the different approaches to deal with uncertainty. The idea of using probabilities to indicate a certain degree of uncertainty is ancient. Nevertheless, due to the large number of factors involved in a problem, it was unmanageable to store and compute all the probabilities necessary to specify the joint probability distribution.

In the eighties different graphical representations driven by probabilistic information appeared. These representations intend to model (in)dependencies between variables. The common aim is avoiding the huge table needed to store the joint probability distribution where every variable has to be related to all the rest. Since some variables are clearly not related to others, this table may present many data that actually are not necessary. From this idea Bayesian belief networks came up. They are normally known as Bayesian networks, whose name comes from Bayes' Theorem used to compute probabilities.

So, the probabilistic expert systems have undergone a great growth lately, and among them, Bayesian networks have become one of the most successful and common choices. However, although a Bayesian network is a good tool for knowledge representation, it is quite complicated to work directly on this structure. Inference over a Bayesian network is usually done over a secondary/auxiliar structure known as junction or join tree. Once this tree is obtained, several probability propagations will be carried out along it. These propagations are useful for both abductive and deductive inference.

Description of the chapters

Chapter 1 will then be devoted to define Bayesian networks (BNs), after having located them in the Expert System environment, and to introduce some of the most standard and known methods for inference which will be referenced along the remaining chapters of this work.

To construct the junction/join tree, the Bayesian network should be processed by a procedure called compilation. This process can normally take a considerable amount of time, which is necessary to minimise. Compilation is divided in a series of operations. Among them, the only *problematical* one is the graph triangulation. Moreover, the resolution of this step presents an NP complexity. For this reason, the first part of this thesis is focused on studying this triangulation process, which is, as explained above, one key point when constructing the tree, and therefore, it will have an important influence on propagation for inference. Chapter 2 will give an insight into this problem first revising some existing techniques and later proposing a new *distributive* approach for triangulation.

It is precisely inference in Bayesian networks the main point we intend to study in this work. Apart from undertaking the triangulation problem itself, we have achieved a great improvement for the compilation in BNs. We are not going to develop a new architecture for BNs inference, but taking some already existing framework for probability propagation such as Hugin or Shenoy and Shafer, we have designed a method that can be successfully applied to get better performance, as the experimental evaluation will show. This method is called Incremental Compilation and will be described in detail through Chapter 3.

In the field of Bayesian networks an Object Oriented *extension* has recently gained relevance. The next part of this thesis will make a study on different approaches to these structures. To clarify the contents we have decided to deal with this subject in two differentiated chapters. In chapter 4 we introduce concepts related to the environment of modular Bayesian frameworks which will be necessary to assimilate when we undertake chapter 5. In this chapter we take the inference problem up again, but for Object Oriented Bayesian Networks (OOBNs).

There have also been some trouble to define correct and fast enough procedures to carry out inference, since the complexity of the structures and *objects* communication maintaining private data makes this process quite different from the "traditional" inference in BNs. We have gone into this subject, studying the various approaches in

the literature, and looking for the cooperation between them in some key aspects. Our proposal to solve some of these arisen problems is the Plug & Play inference technique, which is inspired in other inference techniques applied to the multi-agent paradigm framework called Multiply Sectioned Bayesian Networks (MSBNs). Chapter 5 will show Plug & Play OOBNs method. We will also analyse how this *modular* structures can take advantage of certain techniques described previously in this report.

Finally, chapter 6 completes this work exploring the problem of finding explanations, very useful, for instance, in diagnosis tasks or analysis problems. This is another *kind* of inference that has aroused interest, since the user should not always want to have the posterior probabilities for variables, but an explanation or a set of explanations that could lead to the observed facts. This chapter will go through a new abductive technique based on an explanation tree that attempts to improve the major drawbacks of the main existing algorithms, such as overspecification.

Chapter 1

The role of Bayesian networks and basic concepts about their inference

Man is a reasoning rather than a reasonable animal.
Alexander Hamilton. (1755-1804)
One of the United States' founding fathers

1.1 Expert Systems

Some decades ago certain problems as speech or pattern recognition, determined games such as chess, or even quite complex deterministic or stochastic systems were thought to be solved only by persons, since the formulation and resolution of those problems required specific *skills* exclusively *from human beings* (the ability of thinking, observing, learning, seeing, ...).

Anyhow, the number of achievements obtained in the last three decades by researchers from different fields have shown that many of these problems can actually be formulated and solved by machines.

These endeavours have participated in constructing the wide knowledge field known as *Artificial Intelligence* (**AI**). The expert systems constitute one of the areas that form AI. But, furthermore, most of the other areas inside AI have intersection with the field of expert systems.

There are many distinct definitions for an expert system. We could summarise them like that:

Definition 1 (Expert System [21])
An expert system could be understood as a computerised system (hardware and software)

that emulates human experts in a given specification area. □

It is said that an expert system should be able of processing and memorising information, learning and reasoning in deterministic and uncertain situations, communicate with people and/or other expert systems, take suitable decisions, and explain why these decisions have been taken. It could also be thought as a kind of *advisor* which provides aid to the human experts with a reasonable degree of reliability.

A general scheme for an expert system could be the one shown in figure 1.1. In the left part we can find that there might be some interaction with the user(s). On the right side the expert system is divided into two main components:

- Knowledge. It is normally associated to the so called **Knowledge base** which contains the knowledge about the particular domain that has been previously formalised and structured.

- Reasoning. There is another important module inside an expert system known as **Inference engine**. This is in charge of the operations of search and selection of the elements to be used in the reasoning process and of obtaining answers to the posed queries.

This is a general overview, these two modules can also be divided into smaller components. Besides, some authors consider the working memory as a separate tool.

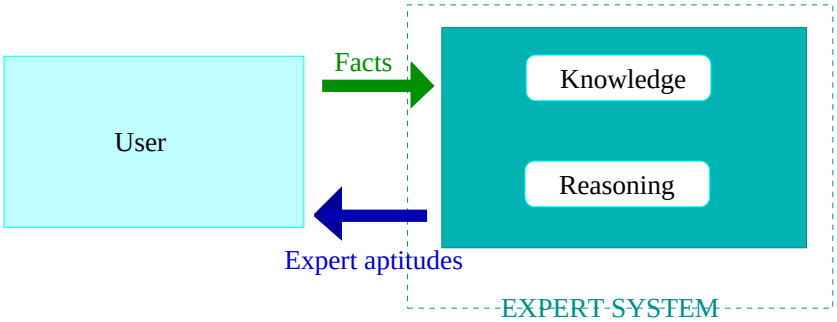


Figure 1.1: An expert system

Many examples of expert systems applications exist in the commercial market. Historically we could remark two of them:

- MYCIN [121] is an expert system for diagnosis, started by Feigenbaum and afterwards developed by E.Shortliffe and his collaborators. Its function is giving advice to doctors in investigation and determination of diagnosis for diseases related to blood infections. MYCIN system, as it is queried by the doctor, first asks general data about the patient: name, age, symptoms, etc. Once this information is known by the system, the expert system gives one hypothesis. After, it first checks the accuracy of the rule premises and then verifies or rejects the given hypothesis. A series of tests have proven that the performance of MYCIN is better than acceptable, when comparing its results to those provided by specialised doctors.

- XCON for eXpert Configurer [8], is an expert system for making network configurations developed by Digital Equipment Corporation. According to the individual requests of the client the computer networks VAX are configured. On account to the wide range of products offered in the market, the complete and correct configuration of such a system is a problem of great complexity.

  Among the functions of this expert system, the answers to the following questions are given:

  1. *Can be the components requested by the client combined in a reasonable and convenient way?*
  2. *Are the system components compatible and complete?*

  These answers are obtained in a very detailed way. XCON is able to check and complete the input requests much faster and more efficiently than the people in charge of that task.

Nowadays, expert systems are used in many and diverse fields. The application of expert systems will be suitable for those cases where the experts have complex knowledge about one clearly delimited area, where no previously established algorithm exists or none of the existing ones can solve some problems.

Another field for application will be those including theories where it is practically impossible to analyse all the theoretically possible cases by means of an algorithm and in a time period that is relatively short and reasonable.

1.2 Treatment of Uncertainty

If we observe the history of expert systems, and particularly of the methods with uncertain reasoning, it is verified that most of the first ones (chronologically) and the most important, have been developed in medicine. However, this uncertainty inherent in the real world problems appears in any other field of the natural science, engineering, law, humanities, ... and more especially in those problems of natural language recognition, both written and spoken, where the implicit information, polysemy, ambiguity and inaccuracy, make indispensable to deal with uncertainty. In fact, this necessity is not only related to expert systems and natural language problems, but also to every branch of artificial intelligence, such as learning, artificial vision, robotics, intelligent interfaces, information retrieval, complex games, etc.

There are essentially three kinds of uncertainty sources [33, 70]: deficiencies in the information, real world features and deficiencies inside the model.

Here we enumerate some of the problems caused by uncertainty:

- Incomplete information. When collecting information about one determined phenomenon which involves several variables, it is quite usual that in some cases such information cannot be found due to various reasons.

- Erroneous information. Within the several phases in which the process of collection information is divided, we can usually find aberrant values, even though many techniques to avoid this are being developed.

- Inaccurate information. Some interest data that are easily expressed by means of natural language, are vague or fuzzy in nature and result too difficult to be expressed in a numerical way.

- Non-deterministic real world. As a consequence of this feature present in the real world, many times from the same causes the obtained effects are different.

- Incomplete model. In many occasions the model we use for approximating to the reality of a certain problem, is incomplete, in the sense that the causes of many phenomena are unknown.

- Inexact model. It could also happen that the model structure is the appropriate one, although the determination of the parameters that rule this model behaviour could have been carried out only in an approximate way.

In summary, the uncertainty treatment is, together with knowledge representation and learning, one of the fundamental issues in artificial intelligence. Therefore almost from the beginning of this field, AI, much attention has been paid to treat uncertainty and many methods have then appeared, motivated by the diverse problems that have been arising.

Methods of uncertain reasoning are classified in two large groups: **numerical methods** and **qualitative methods**. When reasoning is uncertain it is usually done by means of numerical methods and this is normally associated to approximate reasoning (though depending on authors this term can be also related to fuzzy logic or similar models).

Among qualitative methods for treating uncertainty, we could point out those based on non-monotone logics, such as default reasoning models (Reiter's model [103] is probably the most famous), reasoned assumptions systems by Doyle [37] known as Truth Maintenance Systems (TMS) and Assumptions based TMS (ATMS) and the theory of endorsements by Cohen and Grinberg [24, 23]. All these methods work in such a way that when there is no enough information, certain assumptions are done, that can be subsequently revised after the reception of new information. The main problem they present is due to their qualitative nature, so they are not able to consider the different certainty/uncertainty degrees of one hypothesis. They normally lead to an unmanageable combinatory explosion. That is why they are mostly studied because of their theoretical relevance (AI foundations) more than because of the practical applications they could derive.

Regarding numerical methods, among which we find the basis of this thesis, the first attempt was the probabilistic treatment. Bayes and Laplace proposed probability as one measure of personal belief about 200 years ago. In the beginning of 20th century there appeared interpretations of probability as frequencies (long term) associated to situations or experiments that can be repeated; in this line, Fisher statistics' work can be particularly mentioned. But at early 30's, due mainly to the work of L. J. Savage and B. de Finetti, among others, probability as a measure of personal belief is rediscovered.

Another possible classification is based on the differences between intensional and extensional systems, that are better described in Section 1.4. At this point, we will only comment that in this work we will focus on aspects related to an intensional model, more concretely on probabilistic graphical models (Bayesian networks).

1.3 Why Bayesian networks?

Bayesian networks have become of great relevance in the last years. Nowadays many researchers study them and work hard on achieving better efficiency for their processing. But the importance of Bayesian networks is also due to the great number of applications they have provided.

In this section, and before introducing a formal and detailed definition of Bayesian networks, we have considered appropriated to remark those aspects that have made them one of the most used representation formalisms for uncertain knowledge nowadays. We also try to explain the interest in going deeply into their structure as well as how important an improvement of the different algorithms used for their treatment is. Many of these algorithms will be studied in this thesis, and we will attempt to improve some of them.

Bayesian networks have been proved as suitable to model a large number of systems of distinct nature. Some reasons that yield the previous statement are:

- It is a probabilistic model. In order to reflect the uncertainty typical of our environment, it is really important the ability of representing it in a way that, at least, is somehow coherent and understandable. The probabilistic model gives us this feature.

- Bayesian networks give us posterior beliefs given a set of observations. That is, from a set of observable and true facts (this truth could be *softened* in some way) we can obtain some conclusions with a belief degree (probability) associated to them. That is what we call information updating.

- They can be used as a classifier. Classification becomes a hard task when it is related to very complex domains.

- They not only provide one answer to our problem, but also they are able to explain why this answer is right, they can even indicate us which reasoning process they have followed [67].

- They are able to find those variables of greatest impact, those whose relevance factor in the global system is critical.

- There exist many efficient algorithms that work with Bayesian networks even if the number of variables is very high, and throughout the time others are refined

and new methods are developed. This provides a basis for posterior studies and shows a high interest for this representation and processing structure for both scientific and industrial worlds. For that, the technological aspect is also outstanding, since their performance is being improved time and again.

- They get to model the (subjective) knowledge from the expert. In Bayesian networks, knowledge is taken from the experience and knowledge of specialists in a concrete field, trying to produce a faithful representation of it. This leads us to a better communication with experts (model-based representation that is modular and easier to work with).

- They can also be learned from empirical data. At this moment many researchers are working on Bayesian networks learning. There are very good results and a very promising potential regarding this issue.

- It is quite easy to use them. Once the system is modelled, working with Bayesian networks results very intuitive. Its graphical representation enables the visual understanding of the relations among variables and the numerical values that represent the probabilities associated to those relations, which make its management quite simple.

- They offer a representation of uncertain knowledge which is basically more natural than rule-based systems. This makes them easier to be maintained and adapted to different contexts. The possibility of taking decisions about uncertain domains, together with this natural way of connecting elements and simplicity for maintenance make Bayesian networks a very attractive tool, even though very sophisticated methods can be applied.

As we have seen, many interesting features can be found inside Bayesian networks techonology, which have given place to their great popularity within the expert systems world.

A majority of authors coincide with concentrating their main interest in two aspects:

1. Contrary to other models as neural nets, Bayesian networks offer the advantage that an expert can contribute knowledge to the model by means of a causal structure which has a concrete interpretation. On the other hand, when it is learned from data, the obtained representation provides us with a descriptive model/*prototype* of the modelled world.

2. And moreover, the resultant network after being trained is comprehensible and extendible producing probabilities over those variables of interest. This network can be used in a simple way even with arbitrary unknown data (unknown both in training and usage).

1.4 Bayesian networks, causal nets and dependency models

Any system that pursues to imitate the behaviour of a human expert should be able to reason from a knowledge base where most of it consists in sentences and rules that can not be guaranteed as totally true and also containing inaccurate and ambiguous terms in their expression. As a result, this system will have to imitate the procedure of human reasoning, able to obtain valid conclusions in spite of **incompleteness** and presence of **uncertainty** in the available information.

Uncertainty treatment can be approached from different points of view, each one belonging to distinct classifications or taxonomies. A relevant one [99] classifies representation systems and the use of uncertain knowledge in two kinds: *extensional* or syntactic systems and *intensional* or semantic ones.

Extensional systems structure knowledge by means of productions or rules *If - then* and one archetype could be the calculation of certainty degrees used by the expert system MYCIN [121]. In these systems uncertainty is treated as a generalised truth value, where the certainty of a formula is defined as a function of the certainty values of its subformulas. Interdependencies between variables are not considered, uniquely for those variables appearing in a same rule.

Rules for intensional models have a descriptive meaning (instead of a procedural one) and they represent constraints on the current state of knowledge. Some examples of these schemes are the formalism of probability theory and the evidence theory by Dempster-Shafer [112]. In contrast those rules of extensional systems do have a procedural meaning. Hence, the rule $A \xrightarrow{m} B$ implies that being A true, B could be added to the *current knowledge* (sometimes called working memory as well) with a certainty value m, independently of the current state of knowledge. Conversely, for intensional systems the rule $A \xrightarrow{m} B$ says uniquely that the set of worlds in which B is false between those in which A is true has very little likelihood, and for that reason, it should be excluded with probability m. Then, the rule does not let the system do

anything, it just describes a restriction on the current state of knowledge. In Bayesian formalism this rule can be interpreted as "if A is true and A is the **only** known fact, then a probability of m can be assigned to B". If other facts appear in the knowledge base so that A is no longer the only known fact, the permission of assigning probability m to B disappears; unless the knowledge contributed by the new facts is *independent* with respect to B, in this case B can keep probability m.

Therefore, it is meaningful to know the dependencies between those variables involved in the particular knowledge, since otherwise nothing could be deduced. When existing dependencies and all necessary parameters are known, it is possible to act in a procedural way in the framework of Bayesian systems.

1.4.1 Representation of dependency models

At this point we have reasoned the need for expressing (in)dependencies existing among the variables involved in the knowledge domain. Now it is time we saw in which form these (in)dependencies are going to be represented. In view of the fact that human beings are able to detect independencies between variables in an almost intuitive way and without need of calculations, this indicates that the notion of independency should be a primitive concept that would be represented in a qualitative and also explicit way. On the other hand, due to the primitive character of the independency notion, this should have a set of common properties in respect to the distinct formalisms (of quantitative type) with which knowledge can be represented.

Before going on with the corresponding definition, we should mention first the notation that will be used. We will note propositional variables with capital letters without subindex $(A, \ldots, Z)$ or with subindexes that can be numerical or lower case letters $(X_i, X_j, \ldots, X_1, X_2, \ldots)$. The set of variables will be noted with capital letters with subindexes also in capital letters $(X_A, X_B, \ldots)$ and they might also be in italics or calligraphic mode $(\mathcal{V}, \mathcal{A}, \ldots)$. Sometimes we will cite a set of variables as X_D like an n-dimensional variable. Ω_{X_i} represents the set of values (states) that a variable X_i can take. Here we remark that in this work we will always deal with *discrete* Bayesian networks. Likewise, Ω_{X_D} represents the set of values that an n-dimensional variable X_D can take and it is obtained by the cartesian product of Ω_{X_i} for all $X_i \in X_D$. Finally, we will represent by $(\downarrow)$ the *projection* operator: given $X_I \subseteq X_J$ and $x_J \in \Omega_{X_J}$, $x_J^{\downarrow X_I}$ represents a configuration of Ω_{X_I} that is obtained by removing from x_J the coordinate of those variables which do not belong to X_I.

(In)dependency relationships between variables can be formalised using the following framework of abstract nature:

Definition 2 (Dependency model [99])
Let $\mathcal{V}$ be a finite set of variables. A dependency model is defined as a pair $\mathcal{M} = (\mathcal{V}, I)$, where I is a set of rules that assign truth values to the predicate.

$$I(X_I|X_K|X_J)_{\mathcal{M}} \equiv X_I \text{ is independent}^1 \text{ of } X_J \text{ when } X_K \text{ is known,}$$

Being X_I, X_J and X_K disjoint sets of variables in $\mathcal{V}$. □

Intuitively, a set of variables X_I is considered independent of another one X_J, given the values taken by the variables of X_K, if our belief in the values taken by the variables of X_I is not modified when obtaining additional information about the values taken by the variables of X_J. To simplify notation $\mathcal{M}$ is usually omitted from the predicate $I(.|.|.)$.

The problem that appears now is if the knowledge field is broad, the predicate list $I(X_I|X_K|X_J)$ necessary to describe the (in)dependencies between the model variables becomes unmanageable and, hence, it is necessary to find a simpler and more compact representation. Graphical representations seem to be a nice and befitting solution to this problem, which are normally called *dependency graphs*. A dependency graph has a node for each one of the variables in the problem.

Although dependency graphs can be undirected or directed, the latter are the most widely used. The existence of one edge $X \rightarrow Y$ indicates dependency between variables X and Y, but it can also show a causality relationship between them: X is cause of Y. Nevertheless, this statement is not always true, it will depend on the particular case we are attempting to represent.

The topological property that allows us to represent independence statements in directed graphs is the concept of directed separation or just *d-separation* (Pearl [99],Verma y Pearl [126]). Before a formal definition of this concept we will introduce some previous definitions:

Definition 3 (Basic concepts about directed graphs)
Given a directed graph $G = (\mathcal{V}, E)$, we name:

- **path**, *an ordered sequence of nodes $(X_{i_1}, \ldots, X_{i_r})$, so that $\forall j = 1, \ldots, r-1$ either the edge $X_{ij} \rightarrow X_{ij+1} \in E$ or $X_{ij+1} \rightarrow X_{ij} \in E$.*

[1] In fact, we say (marginally) independent when $X_K = \emptyset$ and conditionally independent otherwise.

- **directed path**, *an ordered sequence of nodes* $(X_{i_1}, \ldots, X_{i_r})$, *so that* $\forall j = 1, \ldots, r-1$ *the edge* $X_j \to X_{j+1} \in E$.
- **cycle**, *a path* $(X_{i_1}, \ldots, X_{i_r})$ *in which* $X_{i_1} = X_{i_r}$.
- **directed cycle**, *a directed path* $(X_{i_1}, \ldots, X_{i_r})$ *in which* $X_{i_1} = X_{i_r}$.
- **parents** *of* X_i, *the set of nodes*

$$pa(X_i) = \{X_j \in \mathcal{V} \mid X_j \to X_i \in E\}$$

- **children** *of* X_i, *the set of nodes*

$$chi(X_i) = \{X_j \in \mathcal{V} \mid X_i \to X_j \in E\}$$

- **ancestors** *of* X_i, *the set of nodes*

$$asc(X_i) = \{X_j \in \mathcal{V} \mid \exists \text{ at least one directed path } (X_j, \ldots, X_i)\}$$

- **descendants** *of* X_i, *the set of nodes*

$$des(X_i) = \{X_j \in \mathcal{V} \mid \exists \text{ at least one directed path } (X_i, \ldots, X_j)\}$$

- **adjacent nodes** *to* X_i, *the set of nodes*

$$Adj(X_i) = \{X_j \in \mathcal{V} \mid X_i \to X_j \in E \text{ or } X_j \to X_i \in E\} = pa(X_i) \cup chi(X_i)$$

- **directed acyclic graph (DAG)**, *a directed graph which does not contain any directed cycle.*

□

It is clear that definitions of path, cycle and adjacent nodes are also applicable to undirected graphs.

Definition 4 (Blocked path)
Given a DAG $G = (\mathcal{V}, E)$, *a path* α *between nodes* X_i *and* X_j, *it is said to be* blocked *by a set of nodes* X_Z *if there is a node* $X_k \in \alpha$ *so that satisfies any of the following conditions:*

- $X_k \in X_Z$ *and* X_k *is not a head to head node* $(\to X_k \leftarrow)$ *in* α.

- *X_k is a head to head node in α and either X_k or any of its descendants do not belong to X_Z.*

□

In an undirected graph a path is blocked by a set of nodes X_Z if any node of X_Z is located in the path.

Definition 5 (D-separation)
Given a DAG $G = (\mathcal{V}, E)$ and three subsets of disjoint nodes X_I, X_J and X_Z, it is said that X_I and X_J are d-separated *by X_Z if every path between a node of X_I and a node of X_J is blocked by X_Z. This is denoted by $< X_I|X_Z|X_J >_G$.* □

When the context makes clear the graph we are referring we will omit G in the predicate $< .|.|. >$. The analogous concept to undirected graphs is *separation* or U-separation.

Example 1 In the DAG of figure 1.2 $\{B, D\}$ and $\{C\}$ are d-separated by $\{A\}$, while $\{B, D\}$ and $\{C\}$ are not d-separated by $\{A, E\}$.

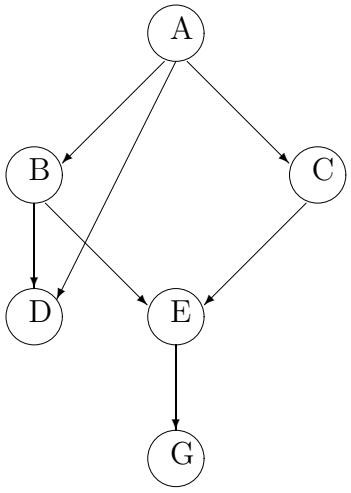


Figure 1.2: Graph example for d-separation

□

Given a dependency model $\mathcal{M} = (\mathcal{V}, I)$ and a dependency graph $G = (\mathcal{V}, E)$ the ideal target would be that the following relation is accomplished:

$$I(X_I|X_K|X_J)_{\mathcal{M}} \iff < X_I|X_K|X_J >_G .$$

This is called a *perfect-map* [99]. However, in practice this situation is not common, usually this relation is at most accomplished in only one direction. That gives place to the concepts listed below (Pearl, [99]).

Definition 6 (I-map)
A DAG $G = (\mathcal{V}, E)$ is said to be an **I-map** *of a dependency model $\mathcal{M}$ if the relation*

$$I(X_I|X_K|X_J)_{\mathcal{M}} \impliedby < X_I|X_K|X_J >_G$$

is accomplished.

□

In an I-map it is guaranteed that all the variables corresponding to nodes that are d-separated are variables independent in the model, but it does not guarantee that those variables corresponding to non d-separated nodes in the graph are variables dependent in the model. We say that a graph G is a *minimal* I-map of $\mathcal{M}$ if when deleting any arc it is not anymore an I-map of the model.

Definition 7 (D-map)
A DAG $G = (\mathcal{V}, E)$ is said to be a **D-map** *of a dependency model $\mathcal{M}$ if the relation*

$$I(X_I|X_K|X_J)_{\mathcal{M}} \implies < X_I|X_K|X_J >_G$$

is accomplished. □

In a D-map the reciprocal situation to the previous one occurs, that is, the independencies of the model are reflected in the graph, but it may be that variables dependent in the model appear as d-separated nodes in the graph.

Within the field of uncertainty treatment of artificial intelligence dependency graphs have become quite popular taking different names, among them we find *influence networks*, *belief networks* or *causal networks or nets*[2]. These names together with *causal*

[2]When causal net is used it is supposed to exist a causality relationship between those variables linked by an edge. Nevertheless, in most cases this causality is not strictly necessary to construct a model, but any weaker notion as a relevance relationship would be enough.

probabilistic networks have been launched and recommended by authors such as Pearl [99], Lauritzen and Spiegelhalter [74] or Neapolitan [87]. In the following, a definition of causal net as a dependency graph, that is, referring to only the qualitative part, is given:

We are going to comment an example of causal net taken from Jensen's book [62].

Example 2 *We wish to model the following system by a causal net:*

> *One morning when leaving home Mr. Holmes realises that the garden grass is wet (H). The reason could be that it has rained during all night or that he forgot switching the sprinkler off (S). At first, his belief on both events increases.*
>
> *Next he sees that in the garden of his neighbour (Mr. Watson), the grass is also wet (W). Now Holmes is almost sure that it has rained during the night and that he effectively switched the sprinkler off.*

Figure 1.3 presents a causal net that models this problem.

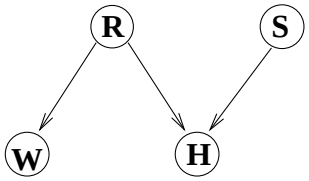


Figure 1.3: Causal net to model the wet grass example

Let us analyse some points about the previous example:

- *If subgraph $W \leftarrow R \rightarrow H$ is considered we can see that the rain is a common cause of the humidity state in both gardens (Mr. Holmes and Mr. Watson gardens). In this case W and H are dependent variables, since if the grass in Mr. Watson's garden is dry we can imagine that it has not rained and, therefore, that the grass in Mr. Holmes garden should also be dry. Then, we have $\neg I(W|\emptyset|H)$. In the network W and H are not d-separated by the empty set and hence the variables are dependent; that is, $\neg < W|\emptyset|H >$. Nonetheless, if we know whether it has rained, knowing the grass status will not add new information on the other grass status. In this case we have the conditional independence $I(W|R|H)$. In the network it is easy to check that it also covers this situation, since $< W|R|H >$.*

- *If we consider the subgraph $R \rightarrow H \leftarrow S$, we see that both the rain and the sprinkler can provoke that the grass in Mr. Holmes's house is wet. It is clear that having rained during the night does not influence the fact that Mr. Holmes could leave the sprinkler on or viceversa and, then, we have the marginal independence $I(R|\emptyset|S)$. This independence is also reflected in the network because $< R|\emptyset|S >$. On the other hand, if the grass status is known the variables become dependent, since if we know that it has not rained during the night and that the grass is wet my belief in forgetting to turn the sprinkler off increases. This kind of reasoning is called "explaining-away". Then, we will have the relationship $I(R|H|S)$. In the network it is easy to see that being H a head-to-head node in the only path between R and S, the nodes are not d-separated by it and, hence, the relationship $\neg < R|H|S >$ occurs.*

- *Finally we want to comment the reasoning process done by Mr. Holmes. When observing that the grass of his garden was wet Mr. Holmes had looked for the causes that could have provoked this effect. Given that there are two possible causes R and S and in absence of any other information Mr. Holmes considers that both could have provoked this situation. But as he sees the grass of Mr. Watson's garden is also wet, this fact reinforces the possibility of rain during the night. Therefore, it increases its belief in the wet grass being the true cause, decreasing his belief in the other possible cause at the same time.*

□

1.4.2 Probabilistic Dependency Model and Bayesian Networks

Until here, we have only referred to the qualitative part of the model, which is the one that allows us to represent the (in)dependencies between the system variables and variables sets. Next, we are going to add to this the quantitative part of the model.

The formalism we will use in this work to represent the quantitative part of the model is the *probability theory*. In a probabilistic environment [28, 49, 75, 122] a probability distribution P can be considered a dependency model using the relation:

$$I(X_I|X_K|X_J) \iff P(x_I|x_K, x_J) = P(x_I|x_K),$$

for every configuration of values x_I, x_J and x_K from the set of variables X_I, X_J and X_K.

There is a set of characteristics to be required for any relation that attempts to catch the intuitive concept of independence. This set is known as *axiomatic set of independence*. The following theorem contains some properties that are always verified by the dependency model associated with a probability distribution:

Theorem 1 (Pearl and Paz [96])
Let us consider the sets of variables X_I, X_J, X_K and X_W which are disjoint two by two. Every probabilistic dependency model $\mathcal{M}$ verifies the following:

A0 *Trivial independence:*
$I(X_I|X_Z|\emptyset)$

A1 *Symmetry:*
$I(X_I|X_K|X_J) \Leftrightarrow I(X_J|X_K|X_I)$

A2 *Decomposition:*
$I(X_I|X_K|X_J \cup X_W) \Rightarrow I(X_I|X_K|X_J)\&I(X_I|X_K|X_W)$

A3 *Weak union:*
$I(X_I|X_K|X_J \cup X_W) \Rightarrow I(X_I|X_K \cup X_W|X_J)$

A4 *Contraction:*
$I(X_I|X_K|X_J)\&I(X_I|X_K \cup X_J|X_W) \Rightarrow I(X_I|X_K|X_J \cup X_W)$

A5 *Intersection:*
$I(X_I|X_K \cup X_W|X_J)\&I(X_I|X_K \cup X_J|X_W) \Rightarrow I(X_I|X_K|X_J \cup X_W)$
This axiom is only verified when the probability distribution P is strictly positive.

□

In a probabilistic environment the quantitative knowledge is usually represented by a joint probability distribution defined over the variables of the system. This creates a problem of representation, considering for example that every variable has two possible states and there are n variables, the number of necessary entries to represent the joint probability distribution is 2^n. It is evident that this number is unmanageable even for small values of n, then for $n = 50$ we will need $2^{50} \simeq 10^{15}$ entries. So, this is needed to represent in another way the joint probability distribution.

If the qualitative part of the knowledge has been represented as a dependency graph G (concretely as an $I - map$ of P) then the quantitative part could be represented in

a more efficient way. Let P be a joint probability distribution over variables $\mathcal{V} = \{X_1, \ldots, X_n\}$. We know that P can be expressed as

$$P(X_1, \ldots, X_n) \;=\; P(X_n|X_{n-1}, \ldots, X_1) \cdot \cdots \cdot P(X_3|X_2, X_1) \cdot P(X_2|X_1) \cdot P(X_1) \quad (1.1)$$

We assume now that the sequence $X_1, \ldots, X_n$ constitutes a topological ordering[3] of the graph G (when G is a DAG it is always possible to find a topological ordering). Let us take any factor $P(X_i|X_{i-1}, \ldots, X_1)$ from the previous expression. Being the sequence of variables ordered topologically with respect to G it is clear that the set $\{X_{i-1}, \ldots, X_1\}$ contains $pa(X_i)$ and does not contain any descendant of X_i, then the following fact is ensured

$$< X_i|pa(X_i)|\{X_{i-1}, \ldots, X_1\} \setminus pa(X_i) >_G .$$

Being G an I-map we will have

$$I(X_i|pa(X_i)|\{X_{i-1}, \ldots, X_1\} \setminus pa(X_i)),$$

which implies that

$$P(X_i|X_{i-1}, \ldots, X_1) \;=\; P(X_i|pa(X_i)).$$

This makes that joint probability distribution can be factorised by means of the next expression, broadly known as the **chain rule**:

$$P(X_1, \ldots, X_n) \;=\; \prod_{i=1}^{n} P(X_i|pa(X_i)) \quad (1.2)$$

Thanks to this, we will only need to store one distribution of conditional probability for every node in the graph. This representation achieves a great saving in the necessary space to store the joint probability distribution. For example, if we suppose that $n = 50$ and that the dependency graph contains 10 root nodes, 10 nodes with one parent, 10 nodes with two parents, 10 nodes with three parents and 10 nodes with four parents, the number of necessary entries to represent the conditional distributions is 620 in contrast to the 2^{50} that would be needed to represent the joint probability distribution. Of course, this factorisation is also of great utility during knowledge acquisition (probabilities).

Once we have seen how to represent both qualitative and quantitative parts of the system, we are going to give a formal definition of *Bayesian network*.

[3]For a sequence $X_1, \ldots, X_n$ of the nodes in a graph to be a topological ordering it is enough if the following is guaranteed: if $X_i \to X_j$ is an edge in the graph, then X_i precedes X_j in the sequence.

Definition 8 (Bayesian network)
Given a probability distribution P over a set of variables $\mathcal{V}$, a Bayesian network (BN) is defined as a directed acyclic graph $G = (\mathcal{V}, E)$ so that

1. *Every node in the graph represents a variable of $\mathcal{V}$.*

2. *G is a minimal I-map of P.*

3. *Every $X_i \in \mathcal{V}$ has a conditional probability distribution $P(X_i|pa(X_i))$ associated to it.*

□

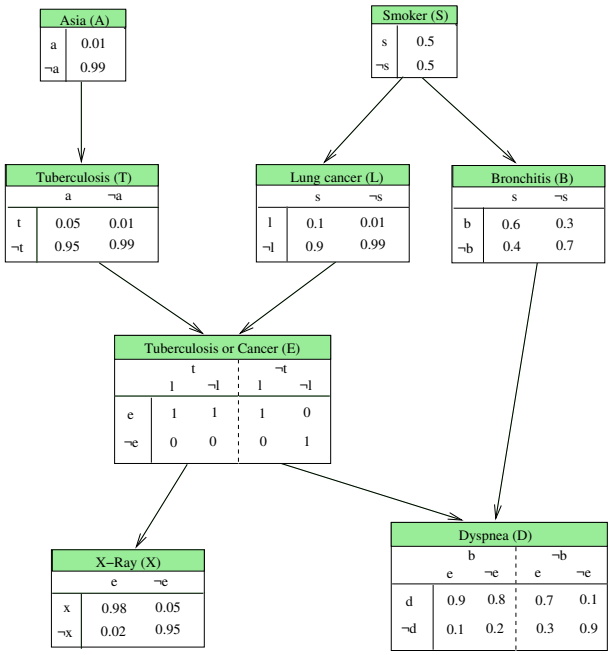


Figure 1.4: Asia network.

Example 3 One example of Bayesian network is the *classical* Asia or chest-clinic domain (see figure 1.4), which was first introduced by Lauritzen and Spiegelhalter [74] and it seeks to model this problem:

Tuberculosis (T) and lung cancer (L) can provoke dyspnea[4](D) and can also be causes of a positive result in a X-ray chest test (X). Also, bronchitis (B) is the other cause of dyspnea. Besides this, the fact of having visited Asia recently (A) increases the probability of suffering from tuberculosis, whereas the fact of being a smoker (S) is one of the possible causes of bronchitis and lung cancer.

In figure 1.4 we not only observe the variables that form the model and the relations among them, which we have just described, but also the numerical values of the associated probabilities. These model the weight of every factor and of the relations among them. For instance, *a priori* (if no evidence is introduced), the fact of having visited Asia is rather improbable, barely 1%. We can see the probability conditional to the parents, $P(.|pa(.))$ as well. For example, the probability of having lung cancer will be greater for a smoker ($S = s$) than for a non-smoker ($S = \neg s$). As we mentioned in section 1.3, it seems clear that the graphical structure of a Bayesian network results very intuitive and working with probabilities in such a way makes easier the insertion of knowledge about the problem by the experts. It should be remarked that in this particular case we are dealing with a network of causal nature. □

Lastly, we want to indicate that obtaining a Bayesian network that gets to model our system is not an easy task. We could say that there are two basic approaches to construct this network, that can be sometimes combined:

- *Manual* construction: in these cases the network is built by a knowledge engineer aided by human experts in the domain s/he tries to model.

- *Automatic* construction : it covers a set of techniques that allow us to build the network from data bases. These methods are known as *learning algorithms* and can be used as a supplement to the knowledge the expert contributes (hybrid learning), or individually. Learning Bayesian networks is a problem that has taken the attention of many researchers in the last years [1, 25, 26, 53, 88]. There have been great advances and many different techniques have lately appeared. However, given the complexity of this task the issue can not at all be considered as closed.

[4]From the Greek word *Dyspnoia* that means difficulty of breathing.

1.5 Inference processes in Bayesian networks

1.5.1 Inference in Bayesian networks

The inference process in Bayesian networks can generally seen as obtaining the posterior probabilities for a set of interest variables $X_I \subset \mathcal{V}$ given an evidence e, that is, we want to obtain:

$$P(X_i|e) \quad \forall X_i \in X_I \tag{1.3}$$

This is normally known as *evidence propagation* or *probability propagation*. We call evidence those facts which are observed and therefore the involved variables are fixed to a certain value. For example, in the network Asia whose probability tables and graph structure was shown in fig. 1.4, if we know certainly that a patient has visited Asia, we can introduce this evidence and then probabilities can be propagated and so updated to their posterior values:

• The evidence to be propagated is $\{A = a\}$, so it is obvious that $P(A = a|A = a) = 1.0 \quad P(A = \neg a|A = a) = 0.0$

• If the interest variables are all the rest, after an inference process, the exact posterior probabilities are:

$$\begin{aligned}
P(B = b|A = a) &= 0.45 \quad P(B = \neg b|A = a) = 0.55 \\
P(S = s|A = a) &= 0.50 \quad P(S = \neg s|A = a) = 0.50 \\
P(T = t|A = a) &= 0.05 \quad P(T = \neg t|A = a) = 0.95 \\
P(E = e|A = a) &= 0.10 \quad P(E = \neg e|A = a) = 0.90 \\
P(L = l|A = a) &= 0.06 \quad P(L = \neg l|A = a) = 0.94 \\
P(D = d|A = a) &= 0.42 \quad P(D = \neg d|A = a) = 0.58 \\
P(X = x|A = a) &= 0.15 \quad P(X = \neg x|A = a) = 0.85
\end{aligned}$$

Many times, what the user seeks is an explanation for that evidence. The interest goes further than the individual conditional probability of each variable and we are also looking for the most likely overall hypothesis or explanation for the current observations. This *abductive inference* is also possible for Bayesian networks, but their working scheme is different from the previous one. There are two types of abductive inference in BNs:

- MPE (Most Probable Explanation) is the most probable configuration of all variables in the BN given evidence. We could call it total abduction as well.

- MAP (Maximum A Posteriori) is the most probable configuration of a subset of variables in the BN given evidence (also called partial abduction).

In general the MPE cannot be found by taking the most probable configuration of nodes individually, and the MAP cannot either be found by taking the projection of the MPE onto the explanation set. Then, total abduction tries to obtain the so-called Most Probable Explanation (MPE).

Definition 9 (Most Probable Explanation) *Let $G = (\mathcal{V}, E)$ be a Bayesian network and x_O one observation of the variables set $X_O \subset \mathcal{V}$. We call $x^* \in \Omega_{\mathcal{V}}$ Most Probable Explanation (MPE) of x_O if*

$$x^* = \arg\max_{\mathcal{V}'} P(\mathcal{V}'|x_O)$$

where $\mathcal{V}' = \mathcal{V} \setminus X_O$ □

The complexity of obtaining this MPE is equivalent to evidence propagation.

We go back to the example of Asia network but this time from the abductive point of view. For instance, we could notice that the patient suffers from Dyspnea, but the question is what might have caused it? After introducing the observation $D = d$ we obtain the configuration (explanation) $\{A = \neg a, B = b, T = \neg t, E = \neg e, L = \neg l, X = \neg x\}$ as the most probable. From here the most important diagnosis is that the disease the patient is suffering from is bronchitis and not tuberculosis or lung cancer and that s/he did not visit Asia. This is only a first approach to illustrate the underlying idea, a deeper study of abduction will be done in chapter 5.

So, we have seen what we want to get from inference processes when we work with Bayesian networks. The next point will be how to obtain this probability values (posterior probabilities used also for MPE). There are many techniques to carry out inference for Bayesian networks. We could first distinguish between two main approaches:

1. Exact methods.- the computed probabilities are exact. It is obviously desirable to have these exact values, but it has also been proven that it is not indispensable for many cases and, on the other hand, quite resource consuming.

2. Approximate methods.- the computed probabilities are close to the exact values, but the way of obtaining them introduces some error. The use of this kind of methods is convenient for large networks or when a speed-up is necessary or a small error is not really important.

Both exact and approximate inference are NP-hard in the worst case.

Secondly, we could think that inference is done directly over the network, but that is not the usual case. In [98] Pearl presented a very efficient way to propagate probabilities in polytrees. Those Bayesian networks whose DAG is a polytree could use this scheme of message passing in order to compute every potential/probability. The problem is that in most real cases a tree is not enough to represent the knowledge domain. Recently, a technique called loopy belief propagation [92] was presented. It uses Pearl's polytree algorithm for Bayesian networks with loops giving place to a very interesting approximate inference.

Nevertheless, what has been broadly used in order to propagate for any kind of network is the clustering method. These methods group variables that are strongly related together. If these groups are organised as a tree, then sophisticated adaptations to the probability propagation can also be done such as Lauritzen & Spiegelhalter method [74], Shenoy & Shafer propagation [113, 114] or Hugin architecture [60]. A more recent technique is Lazy propagation [80, 81]. A more detailed explanation of Shenoy & Shafer method is given below, when the concepts of tree of cliques and join tree are presented. We have picked up this method because it is quite easily understandable, and although there are differences [77] among the distinct techniques, the core in all of them remains the same.

We can find many other methods based on the previous ones that seek a more efficient evidence propagation and whose main feature is the possibility of an approximate inference to improve even more the speed-up, for instance Penniless propagation [19]. There is even a combination of two different techniques as Lazy Propagation with Penniless [20]. In the approximate methods we should also remark the Monte-Carlo algorithms [107, 57].

As we have just introduced, there is normally a secondary structure, which is generally known as join tree (also junction tree), where this inference is performed. This join tree represents in fact another factorisation of the joint probability distribution. Fig. 1.5 illustrates this interpretation of probabilistic expert systems that use Bayesian networks as knowledge representation.

Let us indicate how this join tree is formed and how the probability propagation is performed. As it was said before we have chosen Shenoy & Shafer propagation method for this more detailed description.

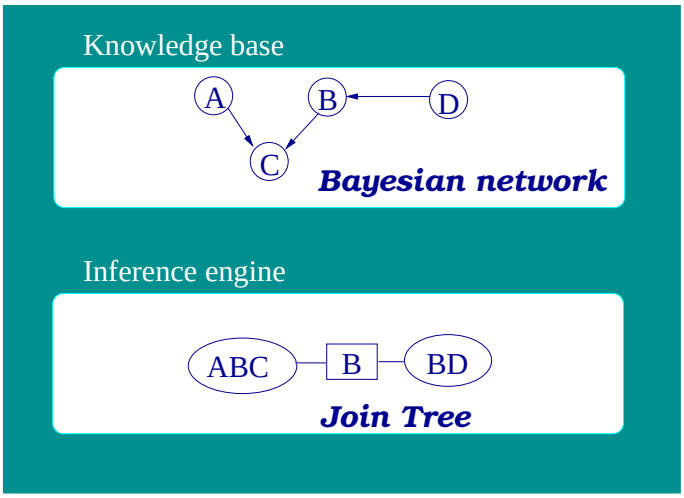


Figure 1.5: A general scheme for a probabilistic expert system.

1.5.2 Join Tree Formation

The tree we want to construct will be our *tool* to compute marginals of a function given as a product of tables on different sets of variables. Let us remind the chain rule (equation 1.2) which is satisfied by a Bayesian network. The clustering made in a join tree tries to put variables in a same *family* together, emulating this factorisation. Every conditional probability $P(X_i|pa(X_i))$ will be necessarily stored and associated to some node of the tree by means of potentials.

Definition 10 (Potential) *A potential ψ defined on a set of variables $X_I \subseteq \mathcal{V}$ will be the mapping $\Omega_I \longrightarrow \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of non-negative real numbers, including the value zero and where Ω_I denotes all the configurations of the values taken by the group of variables variables X_I* □

If $|\Omega_{X_i}|$ is the number of states for variable X_i, there will be $\prod_{i=0} |\Omega_{X_i}|$ elements, and every one will have a real value associated in the potential function. A potential can then be viewed as a function from the space of the possible values for a set of one or more variables to real values. For us, probabilistic information, that is, both conditional (*a priori*) and joint (*a posteriori*) distributions, will be represented using potentials.

For example, in Asia network we can have the *a priori* potential of variable A as $\psi_A = (0.01, 0.99)$, or after introducing the evidence that $S = s$ the probability $P(B|S = s)$ will be represented by the potential $\psi_B =$

	s
b	0.6
$\neg b$	0.4

Let us remark that in this example we have indicated a potential table, but there are other ways to represent a potential function. In chapters 3 and 5 we will treat this point and we will see the benefit of using probability trees [108] and potential trees [19].

But as a tree it is not only a clustering technique, it also establishes links (by branches) between nodes of the tree. These nodes are groups of variables. We remark that a join tree could not be formed by maximal groups, that is, cliques. Then, a tree of cliques will always be a join tree, but the contrary is not always true. The word cluster will also be used for groups of variables. A clique will be a group of variables that form a maximal complete subgraph in the *triangulated* graph G_t, which will be reviewed later when the formation of the tree is described. This configuration of the cliques in the tree will influence directly the propagation method, that is, the way of performing the multiplications to compute the joint probability distribution, and therefore posterior marginals. Those cliques directed connected in the tree will share a separator, which will contain the common variables in both cliques.

Definition 11 (Separator) *Let C_i and C_j be two clusters adjacent in the Join Tree we call S_{ij} their separator and this will be as follows:*

$$S_{ij} = \{V|V \in C_i \cap C_j\}$$

□

These intersection sets are very important for the join tree structure, since the running intersection property must hold for any valid join tree:

Definition 12 (Running Intersection Property) *For every pair of clusters C_1 and C_2 whose intersection is not empty, that is, $V = C_1 \cap C_2 \neq \emptyset$, it is verified that V is contained in all nodes included in the path between C_1 and C_2.* □

It may seem that we have just skipped the definition of junction tree to jump directly to its main components: cliques/clusters, potentials and separators. However, we considered this way more suitable to present the foundations about using a junction

tree. Now, instead of giving a definition of join tree, we will first indicate how it is constructed and how we make the transformation from a Bayesian network BN to one possible join tree JT where inference will take place. This process is called compilation and there are four basic steps:

1. Obtain the *moral graph* from the original DAG that represented the BN.
2. Triangulate this *moral graph*.
3. Identify all the cliques.
4. Connect these cliques in order to form a valid join tree JT.

Let us go through every step explaining what actions they involve and illustrating them for the Asia example. In figure 1.6 we can see the nodes of this network (on the left with whole names, as in figure 1.4, and on the right taking the abbreviations in bracket to make the representation easier).

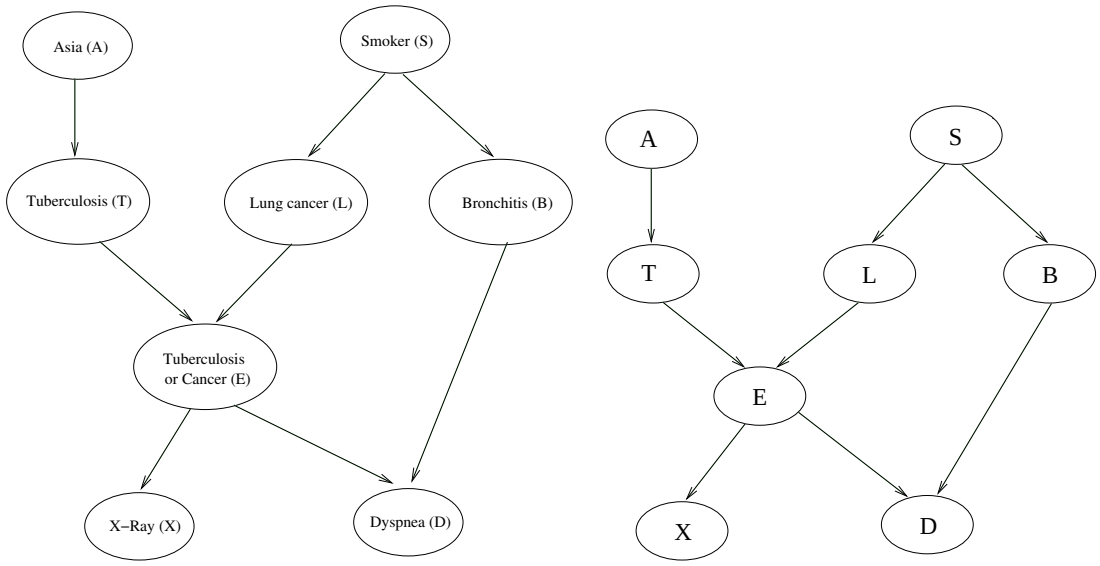


Figure 1.6: The structure of the Directed Acyclic Graph for Asia network. (On the right we show the abbreviated version we will use from here.)

1.- Moralise the graph

So, the first step about moralisation takes the initial DAG G that forms the graphical part of the network and makes it undirected following these two rules: join those

nodes with common parents with a moral link[5], figure1.7.(a), and drop directions of the directed edges, figure 1.7.(b).

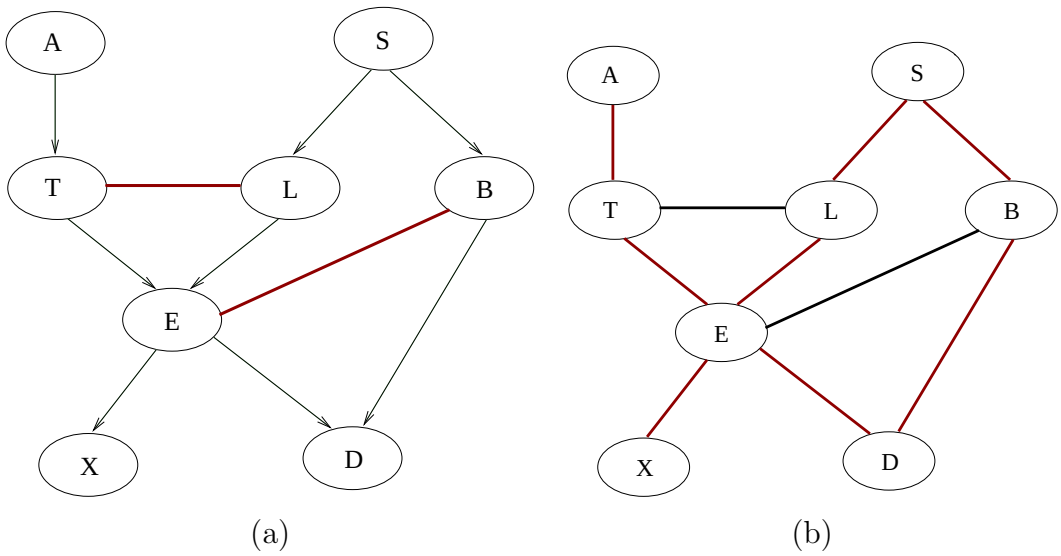


Figure 1.7: Obtaining the moral graph for Asia, indicating the two-steps process.

2.- Triangulate the moral graph

Second phase for compilation is the most problematic step: triangulation, since finding an optimal triangulation is an NP-hard problem [128]. To triangulate a graph it is needed to introduce a *chord* in those cycles of length greater than 3.

Normally, this process is done as the search of a deletion sequence σ which represents an ordering for all nodes in $\mathcal{V}$ (remember that $G = (\mathcal{V}, E)$). Then, σ can also be seen as a function which relates every node $v_i \in \mathcal{V}$ with a unique number between 1 and $|\mathcal{V}|$. Therefore every node will have a position in the deletion sequence. Using this deletion sequence σ the necessary links to add (called *fill-ins*) will be obtained. Thus, the triangulating method will take in this order every node v_i and (1)Remove from the graph all its incident links together with itself, v_i ; (2)Add links between all its neighbours, if they did not exist before. Let us show one example for the moral graph of Asia network (figure 1.8).

In figure 1.8.(a) we proceed to remove the first variable in σ, that is variable A, so we remove its only incident link and itself. In part (b) the same happens with the second variable in the deletion sequence, T, while in part (c) we can see that next

[5]The origin of the term moral comes from *marrying* nodes with common children.

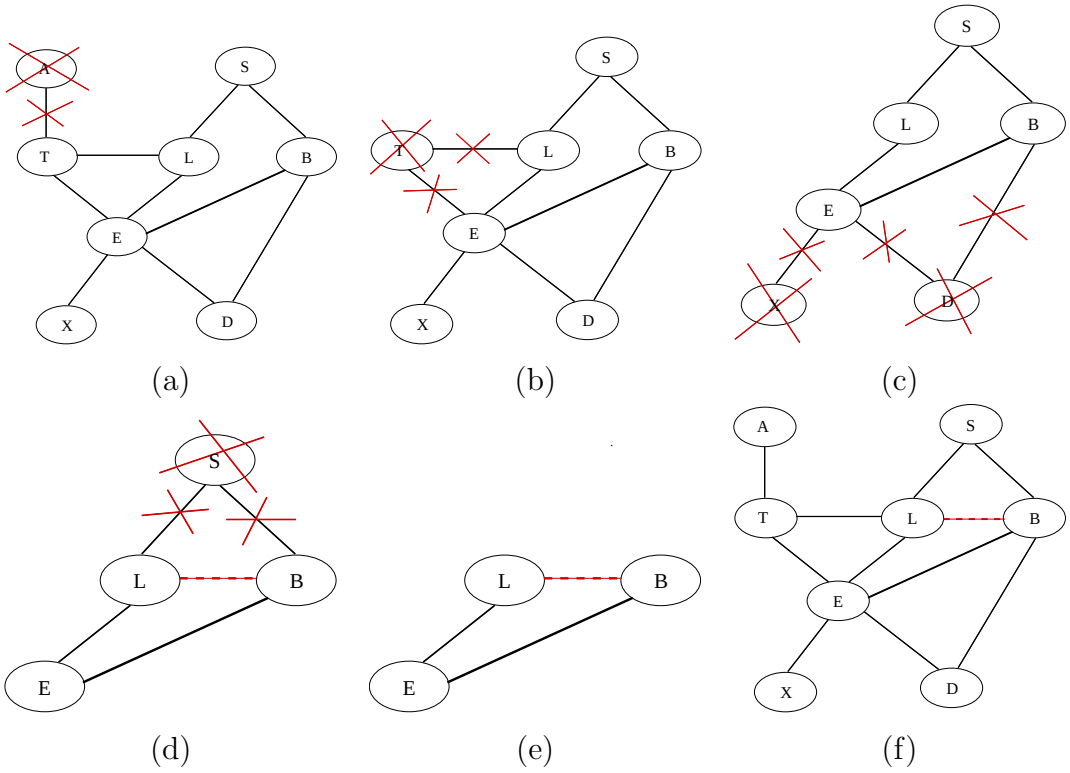
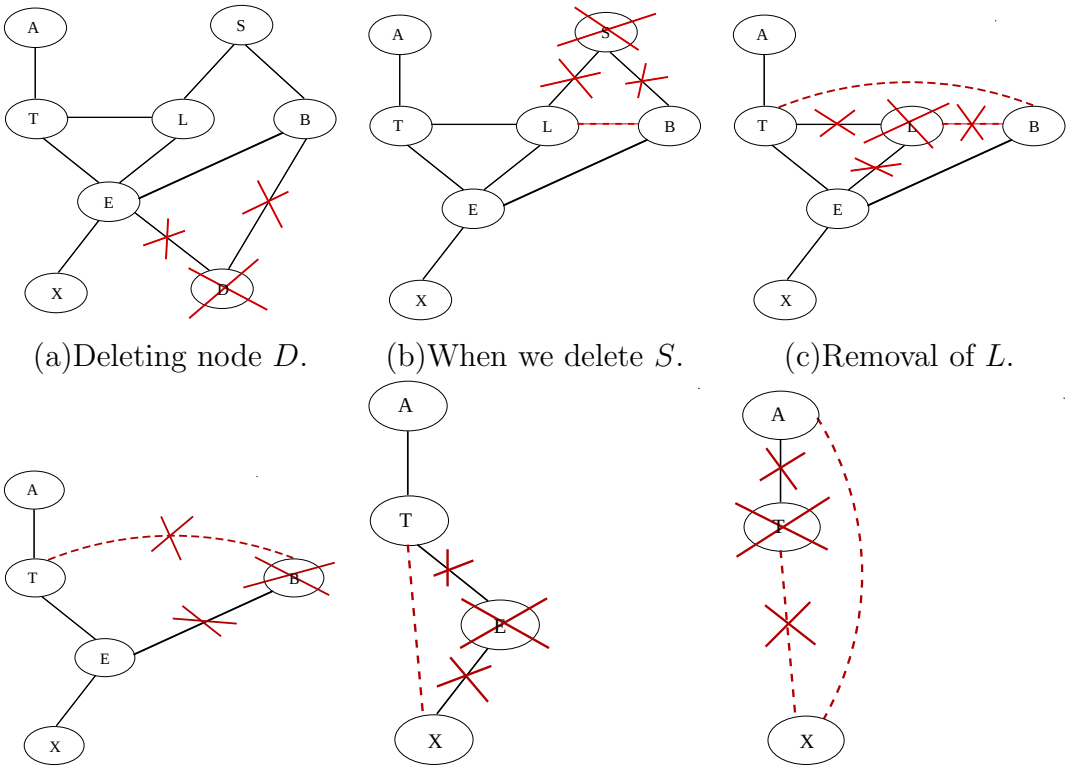


Figure 1.8: Obtaining one possible triangulated graph for Asia. The used deletion sequence is $\sigma = \{A, T, X, D, S, B, L, E\}$.

variables X and D do not produce any fill-in either. It is when the 4-length cycle $S - L - B - E$ is being processed that a chord will in fact be necessary. Since, our sequence is fixed and S is the fifth element, figure 1.8(d) illustrates how this removal of S will imply the addition of a fill-in (triangulating link) between its two neighbours L and B. Finally, in step (e) of the figure we see how the three remaining variables are still connected, so they will not provoke any other fill-in. In fig. 1.8.(f) the resulting triangulated graph is shown.

So, as explained above, triangulation can be viewed as finding the deletion sequence. The method described in the previous paragraph is not complex, but the determination of a good deletion sequence is the most important step. For example, a sequence σ_2 as $\{D, S, L, B, E, T, A, X\}$ would produce the resulting triangulated graph shown in figure 1.10. For a better description of every step, we could look at figure 1.9.



(a)Deleting node D. (b)When we delete S. (c)Removal of L.

(d)Deleting B. (e)When removing E. (f)Removing T.

Figure 1.9: Obtaining another possible triangulated graph for Asia. The deletion sequence used now is σ_2 as $\{D, S, L, B, E, T, A, X\}$.

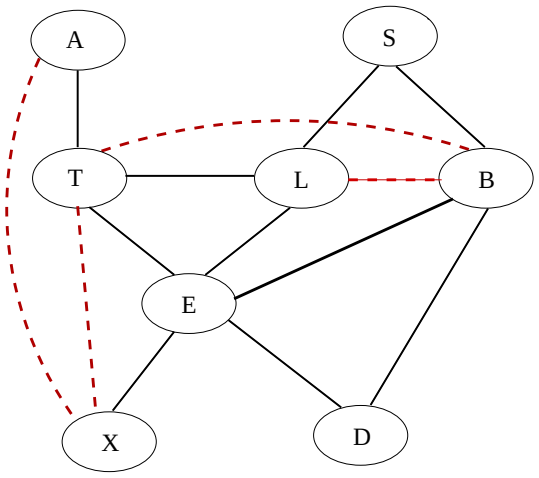


Figure 1.10: Resulting triangulated graph when using deletion sequence $\sigma_2 = \{D, S, L, B, E, T, A, X\}$.

As we can see in figure 1.10 the graph is correctly triangulated, since there are no cycles of length 4 or greater without a chord. However, we have introduced 4 fill-ins instead of the only one needed with σ sequence. That introduces more *unnecessary* relations among nodes that will make a more dense triangulated graph, and will construct bigger clusters. The size of a cluster is crucial for the efficiency of join tree-based algorithms. That will be justified below when the Shenoy-Shafer propagation method will be reviewed. Notice that the triangulation could still introduce many more fill-ins, for example if variable E is the first to be removed, 8 fill-ins in only one step will be introduced!!. And Asia network is a very simple one, since it presents only 8 nodes. It is obvious that the number of possible sequences is equal to all the possible permutations ($|\mathcal{V}|!$), that is, it increases more than exponentially in the number of nodes. So, finding a good triangulation, that is, a good deletion sequence is a very decisive step that will influence the inference processes, and this effect will be more noticeable as the size of the network grows. This fact has motivated that chapter 2 is mainly focused on this issue. A more detailed description of different methods for obtaining deletion sequences as well as new proposals for triangulation will be given in that chapter.

3.- Identify the cliques

Once the graph is triangulated it is time to determine which are the cliques in this triangulated graph. Now we can give a proper definition:

Definition 13 (Clique) *Let G be an undirected graph, then all the maximal complete subgraphs in G are called cliques.* □

In our particular case we will be interested in identifying the cliques corresponding to the triangulated moral graph, G_m^T. As we have already explained, these cliques will be the nodes of the join tree. Since they are extracted from the triangulated moral graph they will also be dependant of the triangulation carried out, that is, of the introduced fill-ins.

Apart from determining the cliques we have to place in a tree-shaped structure. And that leads us directly to the next step.

4.- Build the tree

It implies the establishment of the connections between cliques. From a triangulated graph there can be different possible join trees depending on the clique chosen as root, and sometimes a clique could be connected to different parents.

In order to guarantee that the running intersection property holds (see definition 12), for example we could use the maximum cardinality search [124] for identifying the cliques and then connecting them in a tree. When using maximum cardinality search every node of the network, $v_i \in G$ will have a number associated to it, which indicates the order of cliques in the tree formation. We could also have notated the deletion sequence used for triangulation, and the corresponding clique formed from the elimination of every node v_i, so that cliques will be ordered in the same way ($Clique_i$), as explained in [74]. This second part is only possible if the triangulation has been done through an elimination sequence. Apart from the maximum cardinality search there are other alternative methods of ordering the cliques if no deletion sequence is available. And, on the other hand, it is possible to construct the tree from a deletion sequence if we know the cliques formed when deleting v_i and taking the reverse order of these. Figure 1.11 shows this second procedure for the Asia example with the previous deletion sequence σ.

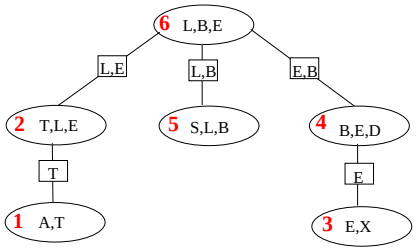
In any case, both methods use he same idea: when identifying the cliques we need to have them ordered in a certain way that will assure the running intersection property. So that, this order will lead to an iterative way of constructing the tree: first clique will be the root, second clique will be joined to this root (their intersection can not be empty unless it comes from a disconnected network), and from there following cliques will be connected to that one among those previously placed in the tree whose intersection (separator) is maximum. If there is more than one, any of them can be chosen (in figure 1.11 the chosen parent is marked in boldface).

i	v_i	$Clique_i$
1	A	$\{A,T\}$
2	T	$\{T,L,E\}$
3	X	$\{E,X\}$
4	D	$\{E,B,D\}$
5	S	$\{S,L,B\}$
6	B	$\{L,B,E\}$
7	L	
8	E	

(a) σ order

i: v_i	$Clique_i$	Sep_i	parents
8: E	-		
7: L	-		
6: B	$\{L,B,E\}$	$\emptyset$	-
5: S	$\{S,L,B\}$	$[L,B]$	**6**
4: D	$\{B,E,D\}$	$[B,E]$	**6**
3: X	$\{E,X\}$	$[E]$	6,**4**
2: T	$\{T,L,E\}$	$[L,E]$	**6**
1: A	$\{A,T\}$	$[T]$	**2**

(b) Inverse order



(c) Resulting join tree

Figure 1.11: Ordering of the cliques from the triangulation $\sigma = \{A,T,X,D,S,B,L,E\}$ in figure 1.8, identification of cliques and tree construction.

It is important to remark the extreme importance of this compilation process due to the previously exposed reasons about the role of the join tree in probability propagations. In this work, we have not only tried to design new techniques for triangulation, but we have also gone a step farther in order to modify the whole process of compilation. Chapter 3 is dedicated to describe this new method that we have named MPSD-based *Incremental Compilation*. The Maximal Prime Subgraph Decomposition (MPSD), which will be properly explained in next chapters, will be a key element for our new methods of both triangulation and compilation. It is time to indicate that chapter 4 deals with exploiting the modular philosophy underlying in MPSD-based Incremental Compilation linking it to Object Oriented frameworks as well as the multi-agent paradigm.

There is still a last and very important point to review: how probabilities are propagated through the tree. As seen before, there are several techniques to perform these operations. In the next point we will detail one of them.

1.5.3 An example of Join Tree Propagation: Shafer and Shenoy Methodology

As explained before a tree of cliques has been thought in order to provide a factorisation of the joint probability distribution. This tree will also serve as basis to operate with these probabilities that will be propagated through the tree.

In this particular example of Shenoy-Shafer architecture, it has been proved that binary trees[6] are the most efficient [115, 116]. Since there is an easy procedure to

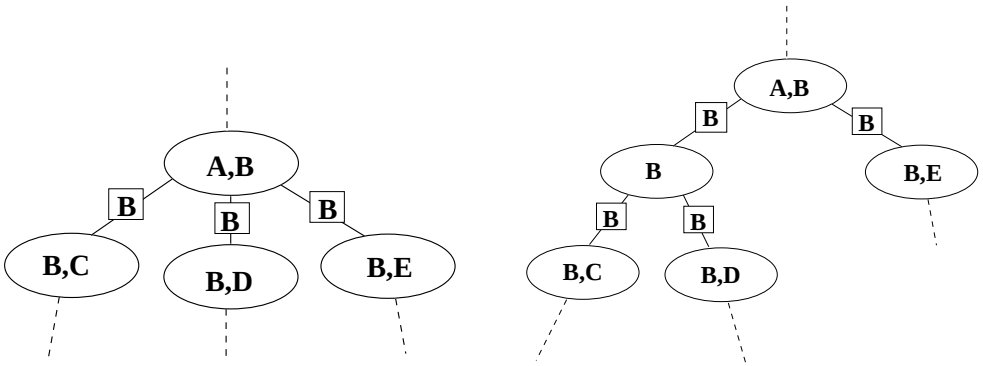
[6]Trees whose nodes have at most three branches: one from its parent and two to its children.

transform a join tree to a binary join tree, that is the first step to explain this technique. So, algorithm 1 basically adds an intermediate node for those with more than three neighbours (impossible in a binary tree), and this new node will maintain the properties of a valid join tree.

Algorithm 1 Converts a join tree to a binary join tree.

```
1: procedure BINARY_TREE(JoinTree 𝒥)
2:     while there is a cluster C in 𝒥 with more than three neighbours do
3:         Let C1 and C2 be neighbours of C.
4:         Create a new node C3 ← (C1 ∪ C2) ∩ C
5:         Connect C3 to C
6:         Disconnect C1 and C2 from C and connect them to C3.
7:     end while
8: end procedure
```

See figure 1.12 as an example of this method, where a new clique is introduced in order to make the tree binary. Notice that 1.11.(c) is initially a binary join tree.



(a) Clique in the initial join tree (b) Clique after applying algorithm 1

Figure 1.12: Example for algorithm 1.

Now, before undertaking the probability propagation, we are going to study how variable potentials (ϕ_V) are assigned to a tree node. We dealt with potentials in section 1.5.2 (see definition 10). But we have deliberately left it out until now, since the way of assigning potentials is dependant to the particular architecture for local computation. In Shenoy-Shafer we have a initial collection of potentials that define the joint distribution. For example, for Asia we would have this set of potentials $\Phi = \{\phi_A, \phi_S, \phi_T, \phi_B, \phi_L, \phi_E, \phi_X, \phi_D\}$, one for every variable in the network. If there is no

observations, then in this set there is a conditional probability $P(X_i|pa(X_i)$ for each node X_i. So, in this case these potentials represent what figure 1.13 shows.

$$\begin{array}{llll} \phi_A = P(A) & \phi_S = P(S) & \phi_T = P(T|A) & \phi_B = P(B|S) \\ \phi_L = P(L|S) & \phi_E = P(E|T,L) & \phi_X = P(X|E) & \phi_D = P(D|B,D) \end{array}$$

Figure 1.13: Potentials associated to Asia's variables.

Every clique will have a potential associated to it. And every separator will present two *mailboxes* that will enable us to send messages between two cliques. The initial potentials ϕ_{X_i} need to be entered in the tree. So, for each one, it is necessary to find a cluster C, containing all the involved variables, that is, $vars(C) \supseteq \{X_i\} \cup pa(X_i)$ and then associate this potential to the corresponding cluster with this *family*. For example, in figure 1.5.3 we have illustrated potentials in the join tree for Asia. Afterwards, the list of potentials is transformed into one potential by multiplication (in the SS case). If there is a clique without an initial potential associated to it, it will have the unitary potential, that is a value 1.0 for every possible configuration of the variables belonging to this cluster.

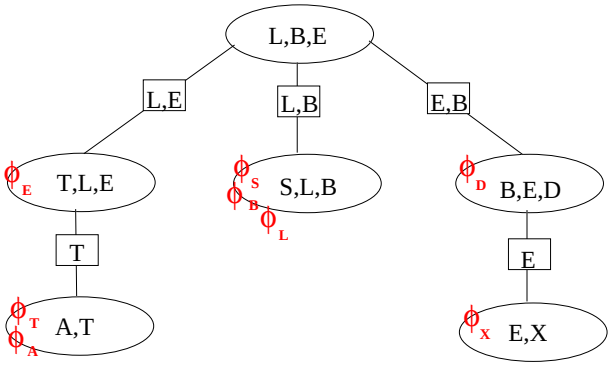


Figure 1.14: Binary join tree with the initial potentials associated to its cliques.

Regarding the *mailboxes* of the separator, they will also be represented as potentials over its set of variables. If separator S_{ij} is located between clique C_i and clique C_j one mailbox will indicate a message in the direction $C_i \rightarrow C_j$ and the other one the opposite direction $C_j \rightarrow C_i$, as figure 1.15 indicates. The message from C_i that is, the upper arrow, will be called C_i-outgoing or C_j-incoming and viceversa. To make

notation clear, we will use μ for potentials in the messages inside a mailbox, for example $\mu_{C_i \to C_j}$ (upper arrow in the figure).

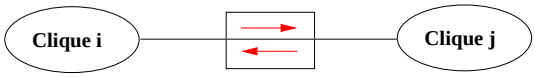


Figure 1.15: Directions of the potential messages in a separator.

Then, this is the information about probabilities that cliques will contain and separators will help to communicate between them. To see the way in which this communication will be done, it is necessary to define a couple of operations over potentials: marginalisation and combination. These two operations are the basic ones for propagating probabilities.

Definition 14 (Marginalisation)
Let X_I and X_J be two sets of variables so that $X_I \subseteq X_J$. Let ψ_{X_J} be a potential on X_J. We obtain the marginalisation of ψ_{X_J} to X_I by means of the result of the following summation:

$$\psi_{X_J}^{\downarrow X_I}(x_I) \;=\; \sum_{x_J^{\downarrow X_I} = x_I} \psi_{X_J}(x_J) \tag{1.4}$$

□

Notice the double use of the operator $\downarrow$: used as a projection when it is applied to configurations of states of variables; and also as the marginalisation operation, as described above.

Definition 15 (Combination)
Let X_I and X_J be two sets of variables. And let ψ_{X_I} and ψ_{X_J} be two associated potentials. Then, the combination of ψ_{X_I} and ψ_{X_J} is a new potential defined over $X_I \cup X_J$ which is obtained by pointwise multiplication:

$$\psi_{X_I \cup X_J}(x) \;=\; \psi_{X_I}(x^{\downarrow X_I}) \otimes \psi_{X_J}(x^{\downarrow X_J}), \quad \forall x \in \Omega_{X_I \cup X_J} \tag{1.5}$$

□

So, initially the probabilities given for every variable given its parents (ψ_{X_i}) is assigned to every clique. The rest have unitary potentials. If one clique $C = \{V_1, V_2, \ldots, V_n\}$

has more than one initial potential associated to it, the potential associated to this clique will be the combination of all of them. That is, if for example the associated potentials are ϕ_{V_1} and ϕ_{V_3}, then $\psi_C = \phi_{V_1} \otimes \phi_{V_3}$. The messages in the separator mailboxes are initially *empty*. Once a message is placed on one mailbox it is said to be *full*.

A node C_i in a join tree can send a message to its neighbour node C_j if and only if all C_i-incoming messages are full except the one from C_j to C_i. Thus, initially leaf nodes are the only capable of sending messages.

The message C_i-outgoing (and C_j-incoming) is computed as:

$$\mu_{C_i \to C_j} = \left\{ \psi_i \cdot \left(\sum_{C_k \in ne(C_i) - C_j} \mu_{C_k \to C_i} \right) \right\}^{\downarrow C_i \cap C_j} \tag{1.6}$$

where ψ_i is the initial probability potential on C_i, $\mu_{C_k \to C_i}$ represent the messages from C_k to C_i and $ne(C_i)$ are the neighbour clusters of C_i. With this scheme one message contains the information coming from one side of the tree and transmits it to the other side. It has been proved [114] that it is always possible to find, at least, one node to send a message until all mailboxes are full. When the message passing ends it is said that the tree is consistent and the following holds for all the nodes in the tree:

$$\psi^m_{C_i} = \psi_i \cdot \left(\prod_{C_k \in ne(C_i)} \mu_{C_k \to C_i} \right) \tag{1.7}$$

where $\psi^m_{C_i}$ is the potential (probability distribution) resulted for the variables in C_i after this propagation. In equation 1.7 we see that for calculating the probability for a set of variables it is necessary to combine the initial potential (ψ_i) with all the incoming messages. It is one of the main differences with other architectures. For example, in Hugin architecture the potentials are directly updated over the clique potential.

The desired probability for a variable X_i can be calculated by marginalising $\psi^m_{C_i}$, where $X_i \in C_i$, over this variable and normalising[7] the result. A probability distribution is normalised when the sum of all probabilities is one ($\sum_{X_I} P(X_i) = 1$). Then, normalisation is the process of transform a non-normalised probability distribution into a normalised one. No matters which clique we choose this probability is the same, for that reason the tree is said to be *consistent* (def. 16).

[7]This step is only necessary when evidence is introduced as we will see later in an example.

Definition 16 (Consistency of the join tree)
A join tree is called consistent after propagating probabilities when $\forall C_k\ X_i \in C_k$ $\mathsf{normalisation}\left(\left(\psi^m_{C_k}\right)^{\downarrow X_i}\right)$ returns the same probability distribution. □

The propagation is usually divided in two stages: upwards and downwards. They are also called collection and distribution. For this scheme it is necessary to select a clique as the *root* one. The first phase (upwards/collection) collects from the root node all the *information* (probabilities) from its neighbours and recursively they collect it from their own neighbours and so, until reaching the leaves of the tree. In the second phase this root clique sends the collected information to its neighbours, which will send theirs recursively until the leaves again.

Shafer-Shenoy algorithm can be written as algorithm 2 shows. Notice that the number of operations performed are depending on the number of cliques and their size. This size is the number of entries for a potential, that is, the product of all the states in the clique variable: if clique C contains a set of variables X_I, then size(C)=$\prod_{i\in I}|\Omega_{X_i}|$. So, as remarked before, a bad triangulation will yield to a less optimal tree, and less efficient propagations. Normally, the *goodness* of a tree is measured in terms of its total space size, which is the sum of all clique sizes $\sum_{C_i}$size(C_i). Sometimes, the biggest clique size is also considered due to its relative importance with respect to the total size.

Algorithm 2 Shenoy-Shafer propagation scheme

1: **procedure** SHENOY-SHAFER(*JoinTree* $\mathcal{J}$)
2: Select a node as *Root*
3: **for all** $C \in Ne(Root)$ **do**
4: UPWARDS(*Root*,*C*)
5: **end for**
6: **for all** $C \in Ne(Root)$ **do**
7: Compute message $\mu_{Root\to C} = \left\{\psi_C \cdot \left(\sum_{C_k\in ne(Root)-C}\mu_{C_k\to Root}\right)\right\}^{\downarrow Root\cap C}$
8: DOWNWARDS(*Root*,*C*)
9: **end for**
10: **end procedure**

where UPWARDS is a procedure that sends a message from leaves to root (see alg. 3) and DOWNWARDS sends messages from root to leaves (see alg. 4).

Algorithm 3 Upwards phase of messages' propagation

procedure UPWARDS(*Cluster* C_1,*Cluster* C_2)
  for all $C \in Ne(C_2) - C_1$ **do** ▷ *C's are every neighbour of C_2 except C_1.*
    UPWARDS(C_2,C)
  end for
  Compute message $\mu_{C_2 \to C_1} = \left\{ \psi_C \cdot \left(\sum_{C_k \in ne(C_2) - C_1} \mu_{C_k \to C_2} \right) \right\}^{\downarrow C_1 \cap C_2}$
  ▷ *When all incoming messages has been received for C_2, $C_2 \to C_1$ can be computed.*
end procedure

Algorithm 4 Downwards phase of messages' propagation

procedure DOWNWARDS(*Cluster* C_1,*Cluster* C_2)
  for all $C \in Ne(C_2) - C_1$ **do** ▷ *C's are every neighbour of C_2 except C_1.*
    Compute message $\mu_{C_2 \to C} = \left\{ \psi_C \cdot \left(\sum_{C_k \in ne(C_2) - C} \mu_{C_k \to C_2} \right) \right\}^{\downarrow C \cap C_2}$
    ▷ *The information is sent from an upper node to its children and so ... until leaves.*
    DOWNWARDS(C_2,C) ▷ *Recursively go downwards.*
  end for
end procedure

Once that the general scheme is presented, we can illustrate Shafer-Shenoy propagation through Asia network and introduce some evidence previously to show the target of the inference process. We can see the graphical illustration of upwards phase indicating at the same time the involved operations in figure 1.17.

Let us suppose we are working with tree in figure 1.5.3, having assigned the indicated potentials and using cluster $[S, L, B]$ as the root. For instance, if we know that a certain patient presents dyspnea, we can introduce this so-called *evidence* or *observation* in no matter which potential containing D, combining it with the clique potential (ψ_i). In this case clique number 4 with variables $\{B, E, D\}$ is the appropriate one. The introduced information (probability) is $P(D = d) = 1.0$ and then $P(D = \neg d) = 0.0$. So, we calculate $\psi_4 = \phi_B \otimes$

d	$\neg d$
1.0	0.0

.

Since we had the initial probability table $P(D|B, E) = \psi_{\{B,D,E\}} =$

	e		$\neg e$	
	b	$\neg b$	b	$\neg b$
d	0.9	0.8	0.7	0.1
$\neg d$	0.1	0.2	0.3	0.9

. And $\psi_{\{B,D,E\}} \otimes$

d	$\neg d$
1.0	0.0

$=$

	e		$\neg e$	
	b	$\neg b$	b	$\neg b$
d	0.9	0.8	0.7	0.1
$\neg d$	0.0	0.0	0.0	0.0

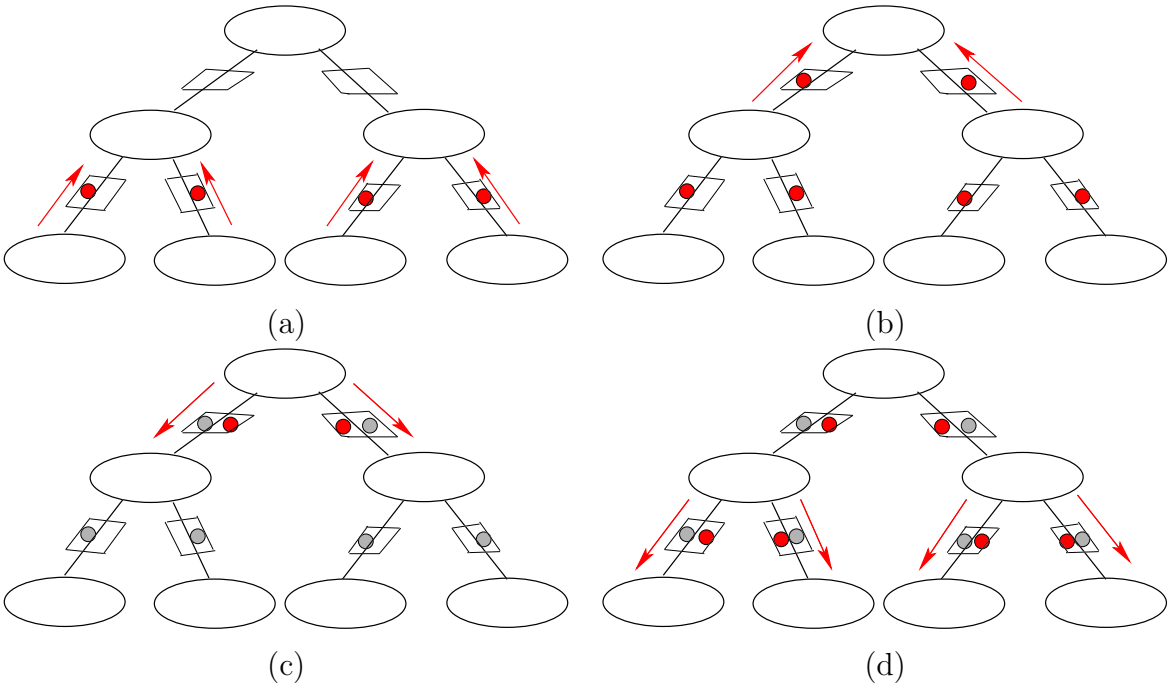


Figure 1.16: Example of upwards (a-b) and downwards stages (c-d)

Then, messages are as follows:

- $\mu_{6\to 4} = \phi_X^{\downarrow E} = 1.0$, that is

d	$\neg d$
1.0	1.0

.

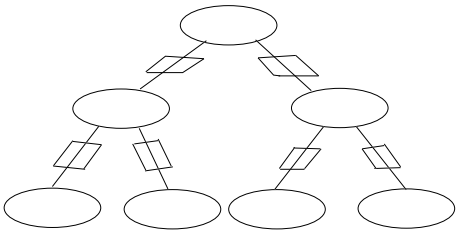
- $\mu_{5\to 3} = (\phi_A \otimes \phi_T)^{\downarrow T} =$

$$\left\{ \begin{array}{|c|c|} \hline a & \neg a \\ \hline 0.01 & 0.99 \\ \hline \end{array} \otimes \begin{array}{|c|cc|} \hline & a & \neg a \\ \hline t & 0.05 & 0.01 \\ \neg t & 0.95 & 0.99 \\ \hline \end{array} \right\}^{\downarrow T} =$$

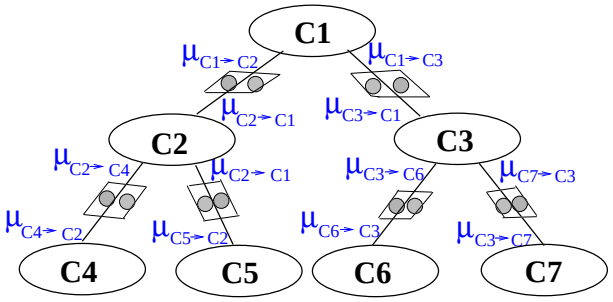
$$= \left\{ \begin{array}{|c|cc|} \hline & a & \neg a \\ \hline t & 0.0005 & 0.0099 \\ \neg t & 0.0095 & 0.9801 \\ \hline \end{array} \right\}^{\downarrow T} = \begin{array}{|c|c|} \hline t & \neg t \\ \hline 0.0104 & 0.9896 \\ \hline \end{array}$$

- $\mu_{4\to 2} = \left[\phi_D \otimes \mu_{6\to 4}\right]^{\downarrow\{E,B\}} =$

$$\left\{ \begin{array}{|c|cc|cc|} \hline & e & & \neg e & \\ \hline & b & \neg b & b & \neg b \\ \hline d & 0.9 & 0.8 & 0.7 & 0.1 \\ \neg d & 0.0 & 0.0 & 0.0 & 0.0 \\ \hline \end{array} \otimes 1.0 \right\}^{\downarrow\{E,B\}} =$$



(a) Initial tree before propagation.



(b) Final (and consistent) tree after propagation.

	e	$\neg e$
b	0.9	0.7
$\neg b$	0.8	0.1

.

- $\mu_{3\to 2}=\left[\phi_E\otimes\mu_{5\to 3}\right]^{\downarrow\{L,E\}}=\left\{\right.$

	t		$\neg t$	
	l	$\neg l$	l	$\neg l$
e	1.0	1.0	1.0	0.0
$\neg e$	0.0	0.0	0.0	1.0

$\otimes$

t	$\neg t$
0.0104	0.9896

$\left.\right\}^{\downarrow\{L,E\}}$

$=$

$\left\{\right.$

	t		$\neg t$	
	l	$\neg l$	l	$\neg l$
e	0.0104	0.0104	0.9896	0.0
$\neg e$	0.0	0.0	0.0	0.9896

$\left.\right\}^{\downarrow\{L,E\}}=$

	l	$\neg l$
e	1.0	0.0104
$\neg e$	0.0	0.9896

.

- $\mu_{2\to 1}=\left[\mu_{3\to 2}\otimes\mu_{4\to 2}\right]^{\downarrow\{L,B\}}=\left\{\right.$

	l	$\neg l$
e	1.0	0.0104
$\neg e$	0.0	0.9896

$\otimes$

	e	$\neg e$
b	0.9	0.7
$\neg b$	0.8	0.1

$\left.\right\}^{\downarrow\{L,B\}}=$

$=\left\{\right.$

	b		$\neg b$	
	l	$\neg l$	l	$\neg l$
e	0.9	0.00936	0.8	0.00832
$\neg e$	0.0	0.69272	0.0	0.09896

$\left.\right\}^{\downarrow\{L,B\}}=$

	l	$\neg l$
b	0.9	0.70208
$\neg b$	0.8	0.10728

.

- $\mu_{1\to 2}=\left[\phi_S\otimes\phi_B\otimes\phi_L\right]^{\downarrow\{L,B\}}=$

$\left\{\right.$

s	$\neg s$
0.5	0.5

$\otimes$

	s	$\neg s$
b	0.6	0.3
$\neg b$	0.4	0.7

$\otimes$

	s	$\neg s$
l	0.1	0.01
$\neg l$	0.9	0.99

$\left.\right\}^{\downarrow\{L,B\}}=$

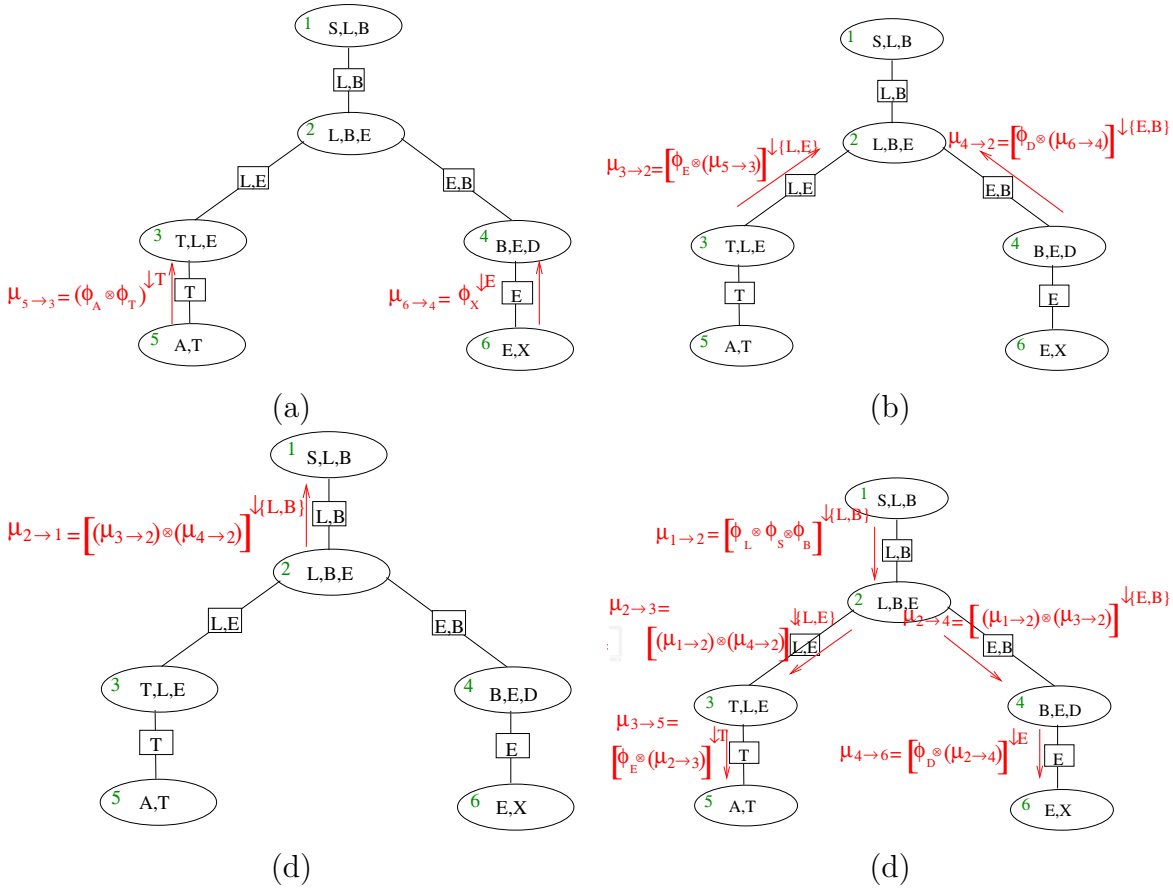


Figure 1.17: Upwards phase for Asia in three steps (a-c) and downwards phase at one sight (d)

$$\left\{ \begin{array}{c|cc|cc} & s & & \neg s & \\ & l & \neg l & l & \neg l \\ \hline b & 0.03 & 0.27 & 0.0015 & 0.1485 \\ \neg b & 0.02 & 0.18 & 0.0035 & 0.3465 \end{array} \right\}^{\downarrow\{L,B\}} = \begin{array}{c|cc} & l & \neg l \\ \hline b & 0.0315 & 0.4185 \\ \neg b & 0.0235 & 0.5265 \end{array} .$$

- $\mu_{2\to 3} = \left[\mu_{1\to 2} \otimes \mu_{4\to 2}\right]^{\downarrow\{L,E\}} = \left\{ \begin{array}{c|cc} & l & \neg l \\ \hline b & 0.0315 & 0.4185 \\ \neg b & 0.0235 & 0.5265 \end{array} \otimes \begin{array}{c|cc} & e & \neg e \\ \hline b & 0.9 & 0.7 \\ \neg b & 0.8 & 0.1 \end{array} \right\}^{\downarrow\{L,E\}}$

  $=$

$$\left\{\begin{array}{c|cc|cc} & e & & \neg e & \\ \hline & l & \neg l & l & \neg l \\ \hline b & 0.02835 & 0.37665 & 0.02205 & 0.29295 \\ \neg b & 0.0188 & 0.4212 & 0.00235 & 0.05265 \end{array}\right\}^{\downarrow\{L,E\}} = \begin{array}{c|cc} & l & \neg l \\ \hline e & 0.04715 & 0.79785 \\ \neg e & 0.0244 & 0.3456 \end{array}.$$

- $\mu_{2\to 4}=\left[\mu_{1\to 2}\otimes\mu_{3\to 2}\right]^{\downarrow\{E,B\}}=$
$$\left\{\begin{array}{c|cc} & l & \neg l \\ \hline b & 0.0315 & 0.4185 \\ \neg b & 0.0235 & 0.5265 \end{array}\otimes\begin{array}{c|cc} & l & \neg l \\ \hline e & 1.0 & 0.0104 \\ \neg e & 0.0 & 0.9896 \end{array}\right\}^{\downarrow\{E,B\}}$$
$$=\left\{\begin{array}{c|cc|cc} & e & & \neg e & \\ \hline & l & \neg l & l & \neg l \\ \hline b & 0.0315 & 0.0043524 & 0.0 & 0.4141476 \\ \neg b & 0.0235 & 0.0054756 & 0.0 & 0.5210244 \end{array}\right\}^{\downarrow\{E,B\}} = \begin{array}{c|cc} & e & \neg e \\ \hline b & 0.0358524 & 0.4141476 \\ \neg b & 0.0289756 & 0.5210244 \end{array}$$
.

- $\mu_{3\to 5}=\left[\phi_E\otimes\mu_{2\to 3}\right]^{\downarrow T}=$
$$\left\{\begin{array}{c|cc|cc} & t & & \neg t & \\ \hline & l & \neg l & l & \neg l \\ \hline e & 1.0 & 1.0 & 1.0 & 0.0 \\ \neg e & 0.0 & 0.0 & 0.0 & 1.0 \end{array}\otimes\begin{array}{c|cc} & l & \neg l \\ \hline e & 0.04715 & 0.79785 \\ \neg e & 0.0244 & 0.3456 \end{array}\right\}^{\downarrow T} =$$
$$\left\{\begin{array}{c|cc|cc} & t & & \neg t & \\ \hline & l & \neg l & l & \neg l \\ \hline e & 0.04715 & 0.79785 & 0.04715 & 0.0 \\ \neg e & 0.0 & 0.0 & 0.0 & 0.3456 \end{array}\right\}^{\downarrow T} = \begin{array}{c|cc} t & \neg t \\ \hline 0.845 & 0.39275 \end{array}.$$

- $\mu_{4\to 6}=\left[\phi_D\otimes\mu_{2\to 4}\right]^{\downarrow E}=$
$$\left\{\begin{array}{c|cc|cc} & e & & \neg e & \\ \hline & b & \neg b & b & \neg b \\ \hline d & 0.9 & 0.8 & 0.7 & 0.1 \\ \neg d & 0.0 & 0.0 & 0.0 & 0.0 \end{array}\otimes\begin{array}{c|cc} & e & \neg e \\ \hline b & 0.0358524 & 0.4141476 \\ \neg b & 0.0289756 & 0.5210244 \end{array}\right\}^{\downarrow E} =$$
$$\left\{\begin{array}{c|cc|cc} & e & & \neg e & \\ \hline & b & \neg b & b & \neg b \\ \hline d & 0.03226716 & 0.02318048 & 0.28990332 & 0.05210244 \\ \neg d & 0.0 & 0.0 & 0.0 & 0.0 \end{array}\right\}^{\downarrow E} = \begin{array}{c|cc} e & \neg e \\ \hline 0.05544765 & 0.34200576 \end{array}$$
.

Then, since all the messages are computed, and propagation has finished, we could calculate the posterior probability of any variables using formula 1.7. For example, for computing $P(B|D=d)$ we could do $\psi_{C_i}^m = \left\{\psi_i \cdot \left(\prod_{C_k\in ne(C_i)}\mu_{C_k\to C_i}\right)\right\}^{\downarrow B} =$

$$\left\{\psi_1 \cdot \left(\prod_{C_k \in ne(C_i)} \mu_{C_k \to C_i}\right)\right\}^{\downarrow B} = \left(\phi_S \otimes \phi_L \otimes \phi_B \otimes \mu_{2\to 1}\right)^{\downarrow B} = \left\{ \begin{array}{c|c} s & \neg s \\ \hline 0.5 & 0.5 \end{array} \otimes \right.$$

$$\begin{array}{c|cc} & s & \neg s \\ \hline b & 0.6 & 0.3 \\ \neg b & 0.4 & 0.7 \end{array} \otimes \begin{array}{c|cc|cc} & b & & \neg b & \\ & l & \neg l & l & \neg l \\ \hline e & 0.9 & 0.00936 & 0.8 & 0.00832 \\ \neg e & 0.0 & 0.69272 & 0.0 & 0.09896 \end{array} \otimes \left. \begin{array}{c|cc} & l & \neg l \\ \hline b & 0.9 & 0.80104 \\ \neg b & 0.1 & 0.19896 \end{array} \right\}^{\downarrow\{L,B\}} =$$

$$\left\{ \begin{array}{c|cc} & l & \neg l \\ \hline b & 0.9 & 0.70208 \\ \neg b & 0.8 & 0.10728 \end{array} \right\}^{\downarrow B} = \left\{ \begin{array}{c|cc|cc} & s & & \neg s & \\ & l & \neg l & l & \neg l \\ \hline b & 0.027 & 0.1895616 & 0.00135 & 0.1485 \\ \neg b & 0.016 & 0.0193104 & 0.00028 & 0.3465 \end{array} \right\}^{\downarrow B} =$$

$\begin{array}{c|c} b & \neg b \\ \hline 0.32217048 & 0.07528292 \end{array}$. And normalising $\begin{array}{c|c} b & \neg b \\ \hline 0.8105868 & 0.189413199 \end{array}$.

To check the consistency of the tree, let us look to another clique containing B, for example clique number 4, and let us compute $P(B|D=d)$ again.

$$\psi_{C_i}^m = \left\{\psi_i \cdot \left(\prod_{C_k \in ne(C_i)} \mu_{C_k \to C_i}\right)\right\}^{\downarrow B} = \left\{\psi_4 \cdot \left(\prod_{C_k \in ne(C_i)} \mu_{C_k \to C_i}\right)\right\}^{\downarrow B} =$$

$$\left(\phi_D \otimes \mu_{2\to 4} \otimes \mu_{6\to 4}\right)^{\downarrow B} =$$

$$\left\{ \begin{array}{c|cc|cc} & e & & \neg e & \\ & b & \neg b & b & \neg b \\ \hline d & 0.9 & 0.8 & 0.7 & 0.1 \\ \neg d & 0.0 & 0.0 & 0.0 & 0.0 \end{array} \otimes \begin{array}{c|cc} & e & \neg e \\ \hline b & 0.0358524 & 0.4141476 \\ \neg b & 0.0289756 & 0.5210244 \end{array} \otimes \begin{array}{c|c} d & \neg d \\ \hline 1.0 & 1.0 \end{array} \right\}^{\downarrow B} =$$

$$\left\{ \begin{array}{c|cc|cc} & e & & \neg e & \\ & b & \neg b & b & \neg b \\ \hline d & 0.03226716 & 0.02318048 & 0.28990332 & 0.05210244 \\ \neg d & 0.0 & 0.0 & 0.0 & 0.0 \end{array} \right\}^{\downarrow B} = \begin{array}{c|c} b & \neg b \\ \hline 0.32217048 & 0.07528292 \end{array}$$

.

We can then check how the two tables on B coincide.

1.5.4 Example of other related techniques: Variable Elimination

We can also find examples of algorithms that apply this clustering method but they do not present explicitly a propagation. Variable elimination is the most representative example in this direction [110, 79, 135]. Many author references are related to [30] which uses the same variable elimination method but enhances it by means of one organisation in buckets and the technique is called as *bucket elimination*.

We are going to present the basic algorithms for the general variable elimination method. The name for this method, Variable Elimination (also VE) cames from the way it is carried out: Variables from a list are summed out one by one. It is necessary to have an ordering ρ by which variables outside $X \cup Y$, being X the interest variable and Y a list of observed variables, has to be summed out.This is called *elimination ordering*. The underlying idea is similar to the elimination sequence for triangulation previously reviewed.

Algorithm 5 Variable Elimination Algorithm

function VARIABLE_ELIMINATION($\mathcal{F}$,$\mathbf{X}$,Y,Y_O,ρ)
▷ $\mathcal{F} \rightarrow$ the list of conditional probabilities in a Bayesian network (potentials combined with the observations).
▷ $X \rightarrow$ a list of target variables.
▷ $Y \rightarrow$ a list of observed variables.
▷ $Y_O \rightarrow$ the corresponding list of observed values, the evidence e is then $Y = Y_O$.
▷ $\rho \rightarrow$ an elimination ordering for variables outside $X \cup Y$.
    for all $y \in Y$ **do** $y \leftarrow y_O$ ▷ *y_O represents the observed value for y, indicated in Y_O.*
    end for
    while $\rho \neq \emptyset$ **do**
        Remove the first variable v from ρ
        SUM-OUT($\mathcal{F}$,v)
    end while
    $h \leftarrow \left(\prod_{f_i \in F} f_i\right)$ ▷ *f_i are all the factors on $\mathcal{F}$ and h will be a function of variables in X.*
    return $\left(\dfrac{h(X)}{\sum_X h(X)}\right)$ ▷ *Renormalisation.*
end function ▷ *The output is $P(X|Y = Y_O)$.*

The algorithm SUM-OUT is also described (see alg. 6). If we have a joint probability $P(v_1, v_2, \ldots, v_n)$, that is, $P(V_1 = v_1, V_2 = v_2, \ldots, V_n = v_n)$, and it is factorised into the multiplication of a list $\mathcal{F}$ of factors, then SUM-OUT($\mathcal{F}$,v_1) returns a list of factors whose multiplication is $P(v_2, \ldots, v_n)$. It could also be expressed as a marginalisation: $\Big[P(V_1 = v_1, V_2 = v_2, \ldots, V_n = v_n)\Big]^{\downarrow\{V_2,\ldots,V_n\}}$.

Algorithm 6 Summing out a list of factors over a variable.

1: **function** SUM-OUT($\mathcal{F}$,V)
   ▷ $\mathcal{F} \rightarrow$ a list of factors.
   ▷ $V \rightarrow$ a variable.
2: $\mathcal{L} \leftarrow \emptyset$ ▷ *$\mathcal{L}$ will contain the partial list of summed-out elements.*
3: **for all** $f_i \in \mathcal{F}$ **do**
4: **if** f_i contains V **then**
5: $\mathcal{F} \leftarrow (\mathcal{F} \setminus f_i)$ ▷ *Remove f_i from $\mathcal{F}$.*
6: $\mathcal{L} \leftarrow (\mathcal{L} \cup f_i)$ ▷ *Add f_i to the list $\mathcal{L}$.*
7: **end if**
8: **end for**
9: $\mathcal{F} \leftarrow \mathcal{F} \cup \{\otimes_{f_i \in \mathcal{L}} f_i\}^{\downarrow V}$ ▷ *Add the new factors to $\mathcal{F}$*
10: **return** $\mathcal{F}$ ▷ *Returns the desired list of factors*
11: **end function**

Notice that given $\mathbf{X}$ as a list/set of interest variables, if we wish to compute $P(x_i|e) \forall x_i \in X$, VE should be executed $|\mathbf{X}|$ times.

The complexity of VE can be measured by the number of numerical multiplications and numerical summations it performs. An *optimal elimination ordering* is one that results in the least complexity. The problem of finding an optimal ordering is NP-complete [3] as it happened in triangulation.

Join Tree Propagation and Variable Elimination are different ways of performing inference. Even if they present some common features each one encodes its particular space-time tradeoff. Which one is more appropriate depends on the problem to treat: which kind of queries will be done, the complexity of the network and other factors. In general VE is more efficient when we have only one *query* (either containing one variable or an n-dimensional list) since the tree construction is avoided and some parts of the network can be discarded. Nevertheless, it becomes more inefficient if we wish to compute all the marginal probabilities.

We have now introduced the science field where Bayesian networks are located, their interest and importance, their definition and how inference is carried out for them. The following chapters will cover distinct aspects of this inference providing algorithms and techniques that get to optimise some inference processes in both inductive and abductive[8] directions.

[8]Inference processes specific for abduction will be reviewed in chapter 5.

Chapter 2

New approaches to the problem of triangulating Bayesian networks

It is indeed wonderful that so simple a figure
as the triangle is so inexhaustible in properties.
How many as yet unknown properties
of other figures may there not be?
August Crelle. (1780–1856)
German civil engineer and mathematician.

2.1 Introduction

As explained in chapter 1 our departure point is the view of a particular probabilistic expert system as *Knowlegde Base* (KB) + *Inference Engine* (IE). So, for us, the knowledge base will correspond to the Bayesian network that models the particular problem and the inference engine tool is given by its associated join tree, which is constructed from the network.

As seen in the previous chapter, compilation is the process of transforming a Bayesian network into this secondary structure. We have also already introduced triangulation as one necessary phase to get a valid join tree. Nevertheless, the resulting join tree from this (compilation) process is not unique for a given BN. Thus, a quite interesting feature is to have the ability of choosing the best tree among all possible ones for a particular BN. When using the join tree for inference, it is clear that the better this tree is the better our inference engine will work. Inference leads to a considerable number of operations and computation, being our target to find a valid tree

for our network, but also as simple as possible[1].

In compilation, the triangulation of Bayesian networks is the hardest task, although it is easy to find a valid triangulation, there is not an efficient[2] algorithm to get the *best* one. Shachter et al. [111] showed that the different exact methods of probabilistic propagation in causal networks, either based on the original DAG or in constraint-based techniques or based on an undirected graph related to the original one or based on node reducing techniques, are all particular cases of the general algorithm called *clustering algorithm*, as mentioned in the first chapter. We should remark that the complexity of a propagation algorithm is exponential in the number of variables of the biggest clique in the tree.

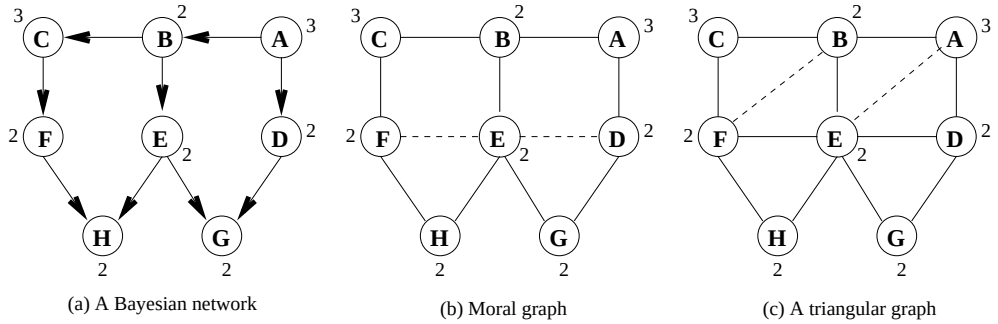
The join tree is obtained directly from a double-step process of (1) moralise the network and (2) triangulate the resulting moral graph. The common point for exact methods is that all build the tree, and they differ in the way they do this construction, each one constructs the tree which is most suitable for its inference operations. That is, each method seeks the best possible triangulation to perform its particular operations. Since posterior inference will be better when the sizes of the formed groups are smaller, getting the best exact method is equivalent to getting the best triangulation in terms of the sizes for the generated groups.

As a result, different join trees can be obtained from the same Bayesian network, but due to the fact that the efficiency of propagation algorithms depends on the complexity of the join tree over which the inference is carried out, it would be desirable to find the *best* possible join tree from the given network. However, being triangulation an NP-hard problem[128] (but of great importance for the efficiency of propagation algorithms), generally, the real goal is to find a *good*[3] join tree, even though this is not the optimal one. Although obtaining this optimal triangulation in the context of Bayesian networks is NP-hard, there exist algorithms getting *good* triangulations using a *reasonable* amount of time, as we will review.

[1] A JT captures (in)dependence relations between variables, but the groups of dependent variables might be bigger than necessary. Comparing two valid join trees, JT_1 and JT_2 related to the same network, the more complex the tree is, the more unnecessary dependencies it is actually including.

[2] Of course, we could always try to test all possible triangulations and pick up the best one, but that will be too expensive computationally speaking.

[3] For us the *goodness* of a join tree will be related to its (state space) size, interpreting this as the number of necessary entries to store the probability tables associated to the join tree nodes (cliques).



(a) A Bayesian network (b) Moral graph (c) A triangular graph

Figure 2.1: Example of a Bayesian network (a), its associated moral graph (b) and a possible triangulation for it (c). Numbers next to each node indicate the number of states for the corresponding variable.

2.2 The problem of triangulation

The usual technique to triangulate a graph is selecting a *deletion/elimination/ removing/triangulation sequence* containing all the nodes in the graph. The method consists of an elimination process, following the sequence order, which will remove all the nodes. If $adj(X_i)$ denotes the set of nodes adjacent to X_i in the undirected graph, then by *deleting* X_i we refer to the process of adding the necessary fill-ins in order to make $X_i \cup adj(X_i)$ a complete subgraph, and subsequently remove it and all its incident edges from the graph. The triangular graph G_T will be the result of adding to the moral graph the set ($\mathcal{F}$) of fill-ins added during the deletion process. That is, if $G_M = (V, E_M)$ is the moral graph, then $G_T = (V, E_M \cup \mathcal{F})$.

In Figure 2.1 we can see, from left to right, a BN, the moral graph, and the triangulated graph obtained by using the deletion sequence $\sigma = (G, H, D, A, E, F, B, C)$. Next section will show this triangulation method algorithmically.

Several approaches [106, 124, 63, 64, 18, 71, 56, 9, 2, 14, 47], most of them basically heuristic, have been proposed to search optimal solutions for this triangulation problem. And subsequently, these algorithms attempt to solve the problem of obtaining a good join tree from a BN, as next section will show. Some of these methods will be commented in the next section, while section 2.3 will present a new technique for triangulation, which is based on the graph decomposition into its maximal prime subgraphs [93] that will let us perform an independent triangulation for every separate subgraph.

2.2.1 Overview of the main existing methods for triangulation

Algorithm 7 Performs the triangulation of an undirected graph.

```
1: function TRIANGULATE_GRAPH(Graph G_M)
   ▷ We assume we wish to triangulate a moral graph G_M.
2:     Sequence ordering σ ← OBTAIN_DELETION_ORDER(V)  ▷ V is the set of nodes in G_M.
   ▷ OBTAIN_DELETION_ORDER gives a permutation of all the nodes (size |V|).
   ▷ σ can be seen as a function where σ(i) is the node in position i within this ordering.
3:     E' ← ∅
4:     for i ← 1 to n  do
5:         V_i ← σ(i)
6:         E'' ← ∅
7:         for all V_j,V_k ∈ Adj(V_i) and j ≠ k do
8:             if {V_j − V_k} ∉ E' ∪ E_M then
9:                 E'' ← E'' ∪ {V_j − V_k}
10:            end if
11:        end for
12:        ADD_LINKS_TO_GRAPH(G_M,E'')          ▷ We add the links in E'' to the graph.
13:        REMOVE_NODE_FROM_GRAPH(G_M,V_i)
   ▷ And we next remove the current and treated node V_i (and its incident links) from the graph.
14:        E' ← E' ∪ E''
15:    end for
16:    Graph G_M^T ← (V,E_M ∪ E')
   ▷ E' corresponds to the set of fill-ins (F). For other algorithms this set will be returned, which is
   a slight modification of the method we will name GET_TRIANGULATION
17:    return G_M^T
18: end function
```

As already studied, to triangulate a graph G the basic technique is usually to add a set of *extra* edges produced by the elimination of nodes in G one by one [63] (see Algorithm 7). A node N will be deleted by adding arcs/links/edges in such a way that nodes adjacent to it (its neighbours) become adjacent two by two, and then a subsequent deletion of its incident links and the node N itself. This procedure does not guarantee that we get an optimal minimum triangulation either in terms of amount of added edges or in terms of the state space size when nodes are chosen randomly. Moreover, on average these measures in random sequences would normally be much larger than those corresponding to a minimum triangulation.

Then, graph triangulation by means of this elimination technique is essentially a problem of determining a sequential order of the nodes that indicates in which chronological order every node will be deleted. We need a specific strategy for the function GET_DELETION_SEQUENCE (line 2 of the alg. 7). Many algorithms to determine this kind of orders have been proposed. Each one will satisfy certain goals specifically sought, but getting away from other (also) interesting goals. In short, there is not a method which results absolutely the best in all the cases. In the next point within this section, we will describe some of the most known triangulation methods classified in two different families (heuristic greedy algorithms and evolutionary algorithms).

2.2.2 Heuristic greedy methods

This group of techniques is characterised by establishing an ordering criterion based on the search rule *"the next node to be deleted is that one minimising f()"* where $f()$ is in function of one or several measures over the set of nodes within the graph $G = (V, E)$[4]. The most used measures [63] are based on:

1. Nodes $i \in V$:
   - Size.- the number of variables: $s(i) = 1$.
   - Weight.- logarithm of the natural size: $w(i) = log_2 c(i)$, where natural size, $c(i) = |\Omega_{X_i}|$. Depending on the author *Weight* is seen directly as $c(i)$[5].
   - Incident.- number of incident arcs in node/variable i within the moral graph: $a(i)$.
2. Groups $C_i \in \mathcal{P}(V)$
   - Size of the group: $V(C_i) = \sum_{j \in C_i} s(j) = \#C_i$. Then it refers to the number of variables in the group (or clique).
   - Weight: $W(C_i) = \sum_{j \in C_i} w(j)$.
     As it happened with the nodes, sometimes this name is used for denoting the natural size: $S(C_i) = \prod_{j \in C_i} c(j)$.

[4] V is the set of nodes/variables in the graph and E are the links/edges between those nodes in G.
[5] And that will be the approach when *minWeight* is referred in this chapter.

- (*Fill-ins*).- number of introduced edges while the triangulation process: $F(C_i) = \frac{V(C_i)*(V(C_i)-1)}{2} - A(i)$

  where $A(i) = \frac{\sum_{V_j \in C_i} a(j)^{\downarrow C_i}}{2}$. That is, the number of edges necessary to make the group complete except those links already belonging to the moral graph, which are counted in $A(i)$. It could also be expressed as $A(i) = E_m^{\downarrow C_i}$.

We should indicate that other authors use the term size also for the *weight* measure. In this work we will try to write clearly which criterion we are referring to.

From these enumerated measures the following ordering criteria appear. We will assume that $C_i = \{X_i\} \cup adj(X_i)$.

- **Minimum size**.- This criterion is based on selecting as the next node to be deleted that one which minimises the function $f(C_i) = S(C_i)$. At each step, it chooses the variable, among those not yet deleted, which produces a clique of minimum size and then this variable is deleted.

As Rose[105] noted minimum size heuristics is fast[6], but it presents some drawbacks:

- It does not produce, in general, a perfect ordering[7] if the graph is already triangulated.

- It does not generally produce minimal triangulations.

- There exist examples for which the produced triangulation is arbitrarily greater than the triangulation obtained by *minimum fill*.

- **Minimum weight**.- This heuristics presents exactly the same advantages and disadvantages as *minimum size*. Note that when all vertices have the same weight both heuristics are identical.

This heuristics gives good results on the whole. It tries to minimise the total sum of the cliques sizes by minimising, at each step, the size of every clique which is being created. This does not guarantee that the total tree size is optimal, since choosing one variable that produces a minimal clique could force us to produce bigger cliques when other variables are deleted later. However, in general, this method provides trees which are relatively manageable.

[6]It can be implemented for a computation time of order $O(n + e')$, where $e' = e + |\mathcal{T}|$, being e the initial links and T those links added during triangulation.

[7]If $G_\# = (\mathcal{V}, \mathrm{E})$ is an ordered graph then # is called a perfect ordering if $T(G_\#) = \emptyset$. All triangulated graphs $G_{\#'} = (\mathcal{V}, \mathrm{E} \cup \mathrm{T}(G_\#))$ have a perfect ordering, since # presents that feature for graph G.

In [18] another particular heuristics based on the same idea arise, but attempting to avoid its weak points. The main underlying idea of these heuristics is that in the moment of deleting a variable it should be sought to minimise the corresponding[8] $S(C_i)$. However, at the same time, the variable and all its corresponding links are deleted, which simplifies the resultant graph. Therefore, what they pursue is that this simplification for the resultant graph could also be taken into account.

Among the several heuristics that Cano and Moral [18] propose in their work, we find this approach called $H2$. This is very similar to *minimum weight*, at each case it chooses the variable X_i, among all the possible variables to be deleted, which minimises $S(i)/|\Omega(X_i)|$. $H2$ calculates the size of the environment of X_i (size of the clique produced when deleting X_i) only with the adjacent nodes to X_i. In this way, not only are the variables deleted with a less complex environment, but also there is a possibility of deleting a variable with a large number of states which leads to less complex cliques in the future formation of the tree.

• **Minimum fill**.- It considers the function $f(C_i) = F(C_i)$. In each case, it chooses the variable, among those not yet deleted, for which its elimination introduces a smaller number of fill-ins. This method presents the advantage of producing a perfect ordering when the graph is triangulated, but provokes the following drawbacks:

- It is slightly slower than the minimum weight heuristics, that is because the adjacency set for every vertex has to be explored regarding arcs.

- In general it does not produce minimal triangulations.

There exist other heuristic techniques which attempt to tackle the problem of graph triangulating. In [56] they are classified in several groups:

1. Heuristics based on the relation between measures for nodes and clusters. They try to establish algebraic relationships between these two types of measures.

2. Heuristics based on measures for clusters and environments of nodes. They define the *k-neighbourhood* of a node by a distance k, which is determined as the minimum number of arcs to go from one node to the other.

3. Compound heuristics. This sort of heuristics can be conceived as a hybridisation where the criterion to be used will vary on the different temporal stages of the triangulation process.

[8] Each deleted variable produces a group of variables, and when this is maximal it will therefore produce a clique.

4. Iterative heuristics. Instead of using a single heuristic criterion to eliminate a node, they can make several iterations (each one with a different measure) in order to decide. They could be of k-iterations, where k could go from 1 (classical approach) until n. 2-iterations methods are studied in [56].

Since the complexity of finding a minimal triangulation grows as $n!$, it is not possible to carry out an exhaustive search directly, except when n is very small. Nevertheless, to construct an elimination order successively and to stop the execution when the total sum of the weights for the cliques (produced until this moment) exceeds the current smallest weight of a complete ordering could be of use to make an exhaustive search even for moderate-size graphs. Being an NP-complete problem, we can not generally expect that a *branch-and-bound* algorithm could find an optimal ordering within certain time limits. That is, the algorithm should finish either when the number of vertices exceeds a certain limit or when the number of the permutations left as discarded increases too slowly. Of course, the initial ordering will have a huge impact on the algorithm success. Thus, a *branch-and-bound* algorithm should be preferably used combining it with another quite faster algorithm (the first would be the last to apply) able of setting a "good" initial ordering for it with the goal of avoiding examining too many useless orderings and also with the goal of minimising the distance to some minimum ordering (we assume that low cost orderings are closer to a minimum one than a high cost ordering).

We could observe that the mentioned heuristics are only one-step lookahead, i.e., they just take into account that node which minimises a certain criterion if this node was deleted in the next step. We could then think of other heuristics able to look further than the next step. Unlike the heuristics above explained, about those looking beyond the next step, there is not much literature. This makes us think that, although they must produce better triangulations than the former, this improvement is not very significant in contrast to the complexity increase.

2.2.3 Methods based on Evolutionary Algorithms

Due to the limitations found with heuristic methods and given that the use of evolutionary algorithms is increasing within the field of computing, many evolutionary variants have arisen trying to deal with the problem of triangulation.

Evolutionary algorithms are techniques for solving combinatory optimisation problems [86]. Among these methods Genetic Algorithms stand out because of their pop-

ularity and their quite broadly extended use [34, 52, 58]. GAs are usually utilised for optimisation problems and specific data structures and functions adapted to the concrete combinatory problem are created. Typically, the form of an evolutionary algorithm is the one indicated in algorithm 8.

Algorithm 8 Scheme for a general evolutionary algorithm.

1: **function** EVOLUTIONARY_ALGORITHM(*popsize*, *Mutation&Crossover*)
▷ *The program will work with population in which every individual represents a possible solution to the problem.* popsize *indicates the number of individuals considered at each iteration.*
▷ *Usually the evolutionary program will need specific parameters:*
- *mutation (or similar) will help to explore new portions of the search space*
- *and crossover (or similar) will mix good solutions (individuals) in order to (try to) get better ones.*
2: $t \leftarrow 0$
3: $Pob(t) \leftarrow$ CREATE_POPULATION() ▷ $Pob(i)$ *is a population with* popsize *individuals (possible solutions) at instant* i *(Initially* $i = 0$*).*
▷ CREATE_POPULATION() *is usually created in a random way.*
4: **for all** Individual $\mathcal{I}_k \in Pob(t)$ **do**
5: $fitness[\mathcal{I}_k] \leftarrow$ EVALUATE($\mathcal{I}_k$) ▷ *If an individual in* $\mathcal{I}_k$ *has already been evaluated, some computations could be saved skipping to redo the same evaluation.*
6: **end for**
7: **if** STOP_CONDITION() **then**
8: **return** BEST($Pob(t)$)
▷ *Return as solution the best individual(s), determined by* fitness.
9: **else**
10: $t \leftarrow (t + 1)$
11: **end if**
12: $PobAux \leftarrow$ APPLY_GENETIC_OPERATORS($Pob(t-1)$)
▷ *As genetic operators we normally mean:* selection + crossover + mutation
13: $Pob(t) \leftarrow$ COMBINE($PopAux$,$Pob(t-1)$)
14: **Goto** line 4
15: **end function**

As said before, this is a general algorithm. The various unspecified points will depend on the particular case. For line 12 we indicate the typically applied procedure (even if this could differ in some specific cases), which is: a selection is done choosing pairs of individuals in the new population, and for every pair there can be or not an

interchange of the information contained in both individuals (if this is the mechanism we refer to crossover operator). The selection of these pairs is done separately according to a certain selection probability (which might be different for each one), and the process takes place, with or without crossover, until all the selected pairs reach to complete all the necessary elements for a new population.

Later on mutation could be applied where possible modifications are done to the individuals in the selected and already crossed population. The point is to make individual alterations (if applied) for some individuals, that is why this step is also called mutation operator. The fact of leaving an individual altered or non-altered is given by a certain mutation probability which is used for all elements in the iteration t.

On the other hand, there are many other unspecified points that will not depend on the method, but on the particular problem, such as the structure of a solution (individual) and the way of evaluating them.

Lately, numerous evolutionary and bioinspired methods have been used for triangulation. Among the most outstanding, we can find:

- **Genetic algorithms** [56, 71]:
  They basically follow the scheme of algorithm 8. When we attempt to solve a problem by means of this type of algorithm, a series of decisions must be taken. Firstly, it is necessary to determine how an individual will be represented. In the case of triangulation, individuals may be the different permutations, which will indicate the elimination order for variables.

  Like other parameters as the population size, mutation or crossover, those works in the literature have found those types and values which better behave for this particular problem. [71] presents crossover and mutation operators adapted to the case of permutations. These were later modified in [47] to get even better results.

Algorithm 9 Scheme for the algorithm *Simulated Annealing*.

function SIMULATED_ANNEALING(Topology, t_0, Ratio, Num_it[])
▷ Topology *refers to the definition of the topology for the search space.*
▷ t_0 *is the initial temperature related to the "cooling" plan, being* $t_0 > 0$.
▷ Ratio *indicates the ratio of temperature decrease.*
▷ Num_it *determines the number of iterations to be carried out for each temperature, being* Num_it*[t] is the number of iterations for temperature t.*
    State $s_0 \leftarrow$ SELECT_STATE(Topology)
    $S_i \leftarrow s_0$
    Temperature $T \leftarrow t_0$
    $\mathsf{cont}_{temp} \leftarrow 0$ ▷ *cont is the counter of the temperature variations (initially set to zero).*
    repeat
        $\mathsf{cont}_{rep} \leftarrow 0$ ▷ *cont is the counter of repetitions.*
        repeat
            State $s_j \leftarrow$ SELECT_STATE(Topology), $s_j \in$ Neighbourhood(s_i)
            $\delta \leftarrow C(j) - C(i)$.
▷ *C() is a cost function, and our method attempts to reach a state in the Topology that minimises this cost.*
            if $\delta < 0$ **then**
                $i \leftarrow j$
            else
                if RANDOM(0,1)$\leq e^{-\delta/T}$ **then**
                    $i \leftarrow j$
                end if
            end if
            $\mathsf{cont}_{rep} \leftarrow (\mathsf{cont}_{rep} + 1)$
        until $\mathrm{cont}_{rep} == Num\_it[\mathrm{t}]$
        $T \leftarrow$ GET_NEXT_TEMP($\mathsf{cont}_{temp}, Ratio$)
        $\mathsf{cont}_{temp} \leftarrow (\mathsf{cont}_{temp} + 1)$
    until STOP_CONDITION() **return** s_i
end function

- **Simulated Annealing**:
  This algorithm is not exactly evolutionary, it might be placed within the framework of the nature-inspired algorithms, as it came from an attempt to "imitate" a behaviour found in nature. Simulated Annealing is based on the physical process called *annealing* by which a liquid is cooled down until it forms a crystalline

solid. During this process, some particles will occasionally move in such a way that the stability for the developing crystal is reduced. Nevertheless, precisely those movements can provoke the substance to reach a status more stable than the resultant when the particles move only on directions which increase stability immediately. Like evolutionary techniques, simulated annealing is a probabilistic method, which initially moves from one solution to another at random; but as time goes on and the simulation progresses, "temperature" decreases, the corresponding crystal is being formed and the system can not be changed in a freer way.

The general procedure [70] can be described by the pseudocode shown in algorithm 9.

In [64] it is studied the behaviour of this technique for a set of networks, where it was tested that this method was able to give good results. However, despite the provided solutions are of a relative quality when tackling NP-hard problems (like triangulation), it is also true that it cannot be guaranteed in general a convergence of this algorithm less than exponential. For that reason, in [64] it was analysed with limited resources, in cases with execution times independent of the space search size and topology.

- **Ant Colony System** [47]:
  None of the previously mentioned methods made use of the heuristic knowledge attached to the problem, which is one of the main features presented by Ant Colony Optimisation, and this helps to guide (and to speed up) the search process.

  ACO algorithms are multi-agent systems in which every agent behaviour is inspired on real ants' behaviour. Particularly, they model the process followed by real ants when they seek the shortest path from the food source to their nest.

  The manner in which the colony constructs the shortest path is based on the ability of ants to deposit and to smell a chemical substance called *pheromone*. In fact, when an ant goes from the nest to the food source and viceversa, it deposits a small amount of pheromone on the ground. The pheromone placed on the path is used for guiding the colony during the search, so that when an ant finds a branch, it must take a decision, and this is actually a probabilistic decision depending on the amount of pheromone that have been deposited in the different branches.

Then, at the beginning of the process, all paths have the same probability to be chosen (because there is no pheromone), but during the continuous action of the colony, the shortest paths start to be more frequently visited, receiving a larger amount of pheromone and therefore becoming more attractive to the subsequent coming ants. On the contrary, the longest paths will be less frequently visited and this, together with the pheromone evaporation process, makes those paths less attractive to the next ants coming. As a result, the final solution comes up from the *collaboration* of every member in a colony.

Even though ACO algorithms were initially used to solve problems related to graph path searches, mainly TSP (Travelling Salesman Problem) [35], in a later work Dorigo and Di Caro [36] introduced a Meta-heuristics based on the ACO approach, which is valid to deal with general optimisation problems if they can be represented as a graph.

Thus, the representation that [47] uses for triangulation is not, as one could expect, the graph associated to the network and the one we intend to triangulate. This is because all possible permutations are not valid as paths within this graph (cycles). This is the reason why they considered the complete graph defined over the network variables, in such a way that it is always possible to reach a node i from a node j for every pair of nodes (i,j). In consequence, there is a graph-form representations equivalent to the one used for the TSP, but in the asymmetrical case, on account of, in general, it is not the same deleting X_i before deleting X_j as in the reverse order. A triangulation-oriented procedure for ACO is shown in alg. 10.

In this technique a reduction step (it will be explained in the next point) is firstly applied [56] with the purpose of reducing the cardinality of the search space. On the other hand, the introduced heuristic knowledge that we mentioned earlier is in function of the nodes that have been deleted before. This is then heuristic information that have to be treated dynamically.

The obtained results turned out to be quite successful, regarding both accuracy and efficiency, comparing it with other known techniques. It is worth mentioning that these experiments did not need a previous study of the network, or refinement of parameters, as it happens for GAs. It presents also the advantage of having an ant-autonomy feature that could make them fit perfectly in a parallel environment with the aim of gaining efficiency.

Algorithm 10 Triangulation-oriented algorithm with an *Ant Colony System*.

```
1: function ANT_COLONY_SYSTEM(Graph G, Ants)
   ▷ G is the graph representation of the problem, so that it contains a set of nodes N.
   ▷ Ants is the set of ants we are going to use in this algorithm.
2:     repeat
3:         for all Ant a_i ∈ Ants do
4:             j ← RANDOM(|N|)
5:             LOCATE_ANT(a_i,N_j)
6:         end for
7:         for all Ant a_i ∈ Ants do
8:             V ← N
9:             repeat
10:                N_j ← CHOOSE_NEXT_NODE(a_i)
   ▷ Notice that here we are taking a combined (probabilistic + heuristic) decision.
11:                V ← V \ N_j
12:                UPDATE_PHEROMONE_LEVELS(local)        ▷ This uses a local rule.
13:            until V == ∅
14:            for all Ant a_i ∈ Ants do
15:                Path[i] ← RETURN_TO_INITIAL_NODE(a_i)
16:            end for
17:        end for
18:        UPDATE_PHEROMONE_LEVELS(global)               ▷ This uses a global rule.
19:    until STOP_CONDITION()
20:    return BEST_PATH(Path[])                          ▷ This will be the shortest path.
21: end function
```

2.2.4 Other techniques relevant for triangulation

Apart from the two previous approaches which are probably the most widely used, other triangulation techniques can be found in the literature, such as divide and conquer techniques based on the concept of *treewidth* (number of variables, minus one, included in the biggest clique in the join tree) [9, 2]. The idea here is to use a different algorithm to triangulate in which the minimum vertex cut method is needed [40]. At each iteration it finds a minimum set of vertices X which being removed from graph G splits it into two disconnected components A and B such that $A \cup B \cup X = V$. This set X is then called the *minimum vertex cut*. This general algorithm proceeds in the two smaller problems $G[A \cup X]$ and $G[B \cup X]$, that is, those subgraphs obtained by projecting G

on $A \cup X$ and $B \cup X$ respectively. And it goes on in this way so that each subgraph is triangulated such that X becomes a clique in it. As we will see MPSD is somehow based on this principle as well.

There exists another method capable of simplifying the triangulation task. In this case, it deals with a process to be performed prior to triangulate with the chosen method. In bibliography we can find it with different names, being *simplicial* (def. 17) the most broadly used. In [56] it is presented as *reduction*, and consists in eliminating all those nodes that, together with their neighbours, form a complete subgraph,i.e., no fill-in has to be added. This part of the network is then already triangulated and deleting them is not going to add any new fill-in. Another approach uses the application of preprocessing rules in order to reduce the graph [14]. In this approach the authors have developed a set of sophisticated *safe reduction rules* to apply onto the graph before triangulation. The results are good, but this complex technique requires, consequently, more computation time.

Definition 17 (Simplicial node)
*Let $G = (V, E)$ be an undirected graph. A node $N \in V$ is said to be **simplicial** if this node N together with its set of neighbours form a complete node set. That is, the projection on the graph G over the adjacency set plus the node N, subgraph $G' = G^{\downarrow\{N \cup Adj(N)\}}$, is a complete graph.* □

On the other hand, it could happen, as MPSD demands, that we want/need a minimal triangulation (def. 18) which basically means that removing any of the added fill-ins in the triangulation for G would not yield a triangulated graph any more.

Definition 18 (Minimal Triangulation)
*If we have a triangulation $\mathcal{F}$ for an undirected graph G, $G_T = (V, E \cup \mathcal{F})$, denoting the set of fill-ins adding during triangulation, $\mathcal{F}$ is said to be **minimal** if $\nexists\ \mathcal{F}'$ so that $\mathcal{F}' \subset \mathcal{F}$ and $\mathcal{F}'$ is a valid triangulation for G. It is equivalent to say that $\forall \mathcal{F}'$, being $\mathcal{F}' \subset \mathcal{F}$ and $\mathcal{F}' \neq \mathcal{F}$, the graph corresponding to $G' = (V, E \cup \mathcal{F}')$ will not be triangulated (presenting cycles of length 4 or greater without chords).* □

There are many methods and studies for getting a minimal triangulation. The most known and first one is lexicographical search, LEX-M[106], providing a way of obtaining directly a minimal triangulation. The method consists of a particularly designed Breadth First Search (BFS), but labelling vertices (nodes) in a lexicographical way, LEX-BFS. LEX-M applies this labelling procedure along paths. More recent studies

and (sometimes more efficient) methods have been designed [101, 54, 13, 12]. Among them, there is a recent successful technique [10] called MCS-M. This is a simplification of LEX-M where cardinality labels are used instead of lexicographical ones. In a analogical way, it applies the cardinality labelling of (neighbour) nodes along the path. MCS-M, as LEX-M, produces a minimal elimination ordering[9]. Even if both techniques could give different orderings it has been proved [127] that they create the same set of triangulations. The LB-triang algorithm [11] is another recent algorithm that computes minimal triangulations with a computation complexity equal to the most efficient methods, and presenting certain properties that could make it especially interesting, such as it can also be implemented as an elimination scheme.

Apart from direct methods, there exist other approaches (as the one in [13]) where the process identifies from a chordal graph the redundant fill-ins, so that eliminating them, a valid triangulation will be transformed into minimal. If $\mathcal{F}$ is a set of fill-ins that make a graph G triangulated, $G = (V, E \cup \mathcal{F})$, these methods identify a set of links $R_{min} \subset \mathcal{F}$, so that $G_{min} = (V, E \cup (\mathcal{F} \setminus \mathcal{R}_{min}))$ is minimally triangulated. The resulting minimal triangulation is therefore $\mathcal{F}_{min} = \mathcal{F} \setminus \mathcal{R}_{min}$.

Among them, we find the method called *recursive thinning* designed by Kjærulf [63, 65]. This method is also called MINT and will be explained in more detailed in next section, since it will be used for the decomposition in MPSs.

Even though we find strong foundations for triangulation on the theory of graphs in literature, it is obvious that triangulation is still a quite open field to optimisation. Proof of that, there is the important number of researchers currently working on this subject, belonging to the domain of probabilistic systems, but also in the area of theory of graphs and graph algorithms.

Although there is no general technique to perform always an optimal triangulation for any graph, there exist attempts to go as closer as possible as the algorithm QuickTree in [120], stated by the authors as the first algorithm that can optimally triangulate graphs with a hundred nodes in a reasonable time frame. In [51] we find a more modern *branch and bound* method, QuickBB with similar purposes. In [31] graph triangulation is interestingly stated and solved as a constraint satisfaction problem. Another an more recent example is found in the commercial tool Hugin[10] where one technique for optimal triangulation has been implemented. This particular method, as indicated in [59] is a combined exact/heuristic method capable of producing an op-

[9] A deletion sequence that provides a minimal triangulation.

[10] http://www.hugin.com

timal triangulation, but only if sufficient computational resources (primarily storage) are available.

Finally, as a remark, several research works have shown that all existing methods for local computation will imply (maybe in a *hidden* way) a triangulation task. Besides, those methods not using a secondary structure like the junction tree either are less efficient or present another problem of NP-hardness [61].

2.3 A new triangulation approach based on the *divide & conquer* methodology

In this section we are going to describe our method *triangulation by re-triangulation* which combines most of the philosophies previously noted. Firstly, as treewidth-oriented techniques, it uses a method for dividing the total graph in smaller components. Olesen and Madsen[93] launched the possibility of applying the Maximal Prime Subgraph (MPS) Decomposition to the problem of triangulation. So, the idea is to retriangulate separately each MPS, since it has been proved to be perfectly valid for the final result. And, secondly, for those portions it will apply some methods of triangulation based on the two main procedures to get a elimination sequence reviewed above. Then, in this work, we have exploited the previous idea by using both greedy heuristic algorithms and stochastic ones (genetic algorithms).

Then, this section is going to describe MPSD first, and after that, our triangulation method will be outlined. With the purpose of studying empirically the utility of applying this MPSD-based retriangulation we have also carried out experiments over 10 real complex networks. The data about the studied networks and a clear description of the experiments will be given next. Finally, this section will be ended with an analysis from the results and observed features.

2.3.1 Maximal Prime Subgraph Decomposition

The decomposition of an undirected graph has been used as a tool for different tasks performed normally on this kind of graphs, as the triangulation procedure, but also many others. When the decomposition obtains a set of solvable subgraphs, this is a suitable tool for *divide and conquer* algorithms, getting a global solution as the sum of local solutions for smaller and independent graphs.

In particular, we will use the *maximal prime subgraphs* (MPS) decomposition[11] of an undirected graph as an intermediate step in our new approach for triangulation. This idea, already proposed as a possible application of the MPS decomposition by Olesen and Madsen [93], consists of working separately on different parts of the initial graph. In our case, the task to do separately will be the triangulation for each graph. But the Decomposition using Maximal Prime Subgraphs (MPSD) has been used to solve other problems related to graphs such as identification of maximum cliques [125]. In [93] the MPSD is used to construct a tree of MPSs, which is used as a graphical structure over which probabilistic inference is performed by *lazy propagation* [80].

Let us just formalise the concept of maximal prime subgraph, for that, we also introduce the definition for decomposition (def. 19) of a graph and the characteristic for a graph of being decomposable (def. 20). Both of them can be easily related from previously presented ideas, since for constructing the JT we have made some kind of decomposition (MPS Tree will be the one which accomplishes the complete separators condition) whereas triangulated graphs are guaranteed to be decomposable [73] and that is somehow the justification for the necessity for a triangulation step. That is the reason why from here, we will refer to a decomposable graph as a triangulated graph.

Definition 19 (Graph decomposition)
Let $G = (V, E)$ be an undirected graph, and let A and B be two sets of vertices in G, G can be decomposed in A and B if and only if the following conditions are satisfied:

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

□

Definition 20 (Decomposable graph)
If a graph G and its subgraphs can be decomposed recursively until all the subgraphs are complete, then the graph is decomposable[12]. □

[11] Also known as decomposition by clique separators.

[12] Note that a graph can be decomposed without being decomposable.

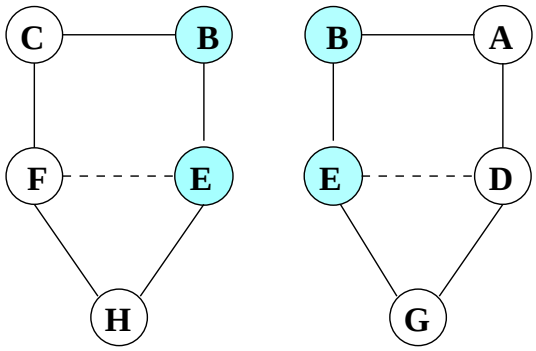


Figure 2.2: A simple example of graph decomposition where $\{B, E\}$ is the clique separator for the BN in fig. 2.1.a.

Then, it is said that a graph is *reducible* if it can be decomposed, that is, its set of nodes contains a clique separator, otherwise the graph is said to be *irreducible/**prime**/ non-separable*. And this leads directly to def. 21:

Definition 21 (Maximal Prime Subgraph)
A subgraph $G(A) = (V, E)^{\downarrow A}$ of a graph G is a Maximal Prime Subgraph of G if $G(A)$ is irreducible and $G(B)$ is not irreducible $\forall B$ so that $A \subset B \subseteq V$.. □

Finally, from the previous concepts it just remains to indicate what the [13] Maximal Prime Subgraph Decomposition is:

Definition 22 (Maximal Prime Subgraph Decomposition)
Let $G = (V, E)$ be an undirected graph. Its Maximal Prime Subgraph Decomposition is the set of induced maximal prime subgraphs of G resulting from a recursive decomposition of G. □

Although other methods have been proposed to obtain the MPSD of an undirected graph [124, 125, 76], the one presented by Olesen and Madsen [93] is especially interesting for us, since it is based on the join tree constructed from a BN (see line 5 of alg. 11 and the first input parameter of alg. 12). The decomposition of the graph in MPSs is returned in a form of a tree, that we will call MPST Maximal Prime Subgraph Decomposition Tree. A MPST, sometimes denoted as $\mathcal{T}_{\mathcal{MPD}}$, is a junction tree for

[13] It can be proved that this decomposition is unique for an undirected graph, as it is the moral graph.

the MPSD of graph G where the clusters will be the maximal prime subgraphs of the corresponding G.

The process to get this particular tree is indicated in alg. 11 and 12. A deeper sight into the relationship MPST and JT will be given (and necessary for the motivation) when explaining Incremental Compilation, in chapter 4. But we should now point out [93] that the tree of MPSs can be seen as an intermediate structure that is located in between the moral graph and the triangulated graph, and so this can be useful for both theoretical algorithms and implementation (data structures).

Algorithm 11 Obtain the MPS Tree or $\mathcal{T}_{\mathcal{MPD}}$ from a graph.

1: **function** OBTAIN_MPSTREE(Graph G)
▷ *We assume this graph G is the graphical part for a Bayesian network $BN = (G, P)$.*
2: $G_M \leftarrow$ MORALISE_GRAPH(G)
3: Triangulation $T \leftarrow$ TRIANGULATE_GRAPH(G)
▷ *G_M^T will be the triangulated moral graph.*
4: $T_{min} \leftarrow$ MAKE_MINIMAL_TRIANGULATION(T)
▷ *In our case we will use the algorithm called* recursive thinning.
5: $\mathcal{T}_{min} \leftarrow$ CONSTRUCT_JOIN_TREE($G_M^{T_{min}}$)
▷ *This can be done for instance by means of the algorithms shown in chapter 1.*
6: **return** CONSTRUCT_MPS_TREE($\mathcal{T}_{min}$,G_M) ▷ *Alg. 12 shows this process.*
7: **end function**

Algorithm 12 Obtain the MPS Tree or $\mathcal{T}_{\mathcal{MPD}}$ from a Junction Tree.

1: **function** CONSTRUCT_MPS_TREE(Junction Tree $\mathcal{T}_{min}$,G_M)
▷ *Junction Tree $\mathcal{T}_{min}$ must have been obtained by a minimal triangulation.*
▷ *G_M is the corresponding moral graph to the network.*
2: $\mathcal{T}' \leftarrow \mathcal{T}_{min}$
3: **repeat**
4: **for all** Separator $S \in \mathcal{T}'$, S connects clusters C_1 and C_2 **do**
5: **if** $\neg$(COMPLETE_GRAPH($G_M^{\downarrow S}$)) **then**
6: AGGREGATE($C1$,$C2$,$\mathcal{T}'$)
7: **end if**
8: **end for**
9: **until** $\forall S_k \in \mathcal{T}'$, S_k is complete ▷ *Until All separators S in $\mathcal{T}'$ are complete for G_M.*
10: **return** $\mathcal{T}'$
11: **end function**

Finally, and since this is the method we have used to guarantee that triangulations are minimal (def. 18), we are going to present the algorithm by Kjærulff [63] that is able of detecting those redundant fill-ins (if any) in a triangulation (see alg. 13). We have used the so-called MINT alg. in [65], but this author has designed another algorithm, FMINT, which is said to be more efficient. Then, eliminating these identified fill-ins that avoid the triangulation T being minimal, we can convert it into a minimal one T_{min}. It is worth commenting that we needed a method for obtaining a minimal triangulation of this kind, but not one finding directly a T_{min}. The reason is that we pursue to compare the behaviour of various triangulation methods, which do not necessarily produce a minimal triangulation, among them and also to compare the same triangulatiom method in the two phases of our designed algorithm RE-TRIANGULATION.

Algorithm 13 Obtain the minimal triangulation corresponding to an arbitrary one.

1: **function** RECURSIVE_THINNING(T,G_m^T,R)
▷ *Both T and R are triangulations in the form of a set of fill-ins.*
▷ *T is the input triangulation we wish to reduce to a minimal one if it is not.*
▷ *R will keep those fill-ins that remain to be tested as redundant or not. For the initial call R is set equal to T, $R \leftarrow T$.*

2: $R' \leftarrow \{e_1 \in T | \exists e_2 \in R : e_1 \cap e_2 \neq \emptyset\}$
▷ *R' will contain those edges of T so that they have non-empty intersection with some other edge of R.*
3: $T' \leftarrow \{\{X - Y\} \in R' | \{adj(X) \cap adj(Y)\}^{\downarrow V}$ is complete $\}$
▷ *T' will contain those edges edges {X-Y} from R' so that the their adjacent neighbour sets intersect into a complete subgraph in G.*
4: **if** $T' \neq \emptyset$ **then**
5: **return** RECURSIVE_THINNING($T \setminus T'$,$G = (V, E \cup (T \setminus T'))$,$T'$)
6: **else**
7: **return** T
8: **end if**
9: **end function**

To see how *recursive thinning* proceeds, let us check a triangulation we know that is not minimal using again the same network. In fig. 2.3 where the initial triangulation T is clearly not minimal, since there exist $T' \subset T$ which still remains as a triangulation. This figure illustrates the (two) different recursive calls to the alg. 13 for this case, indicating the value of variables and parameters. Finally, it returns the corresponding

T_{min} where unnecessary fill-ins has been identified and removed. Therefore, line 4 of alg. 11 is solved in a satisfactory and proven to be correct [65].

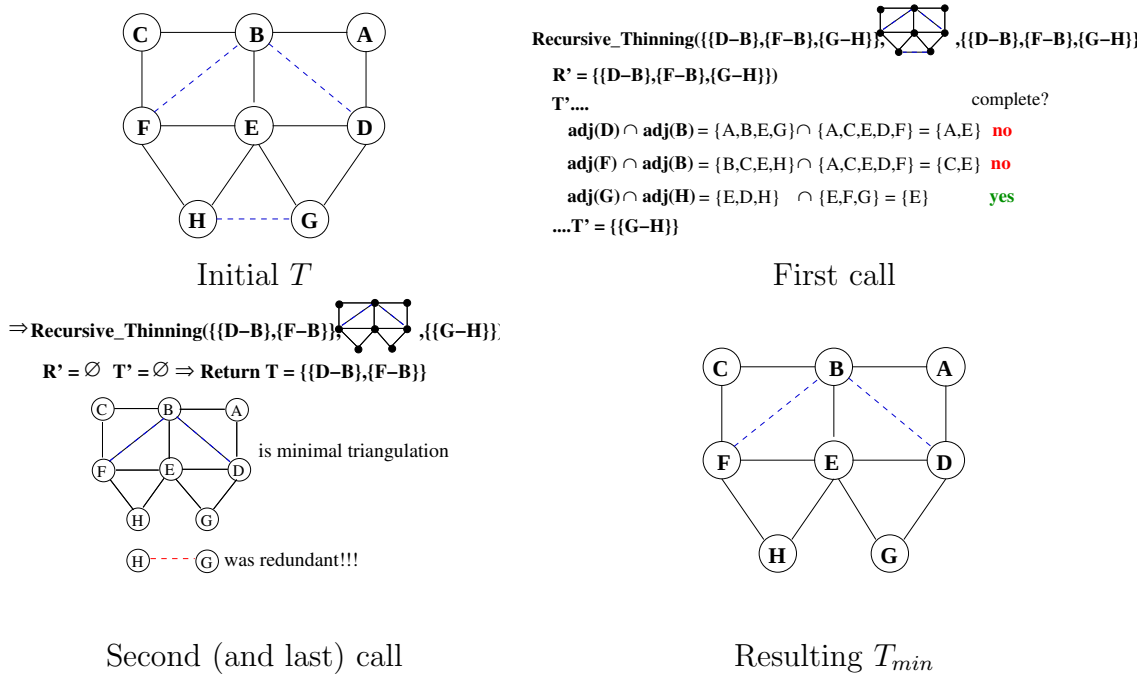


Figure 2.3: Application of recursive thinning to a simple example. It is clear that triangulation T is still a valid triangulation without $\{H - G\}$. We see here how the algorithm detects that. In this particular and easy example only two calls of RECURSIVE_THINNING are necessary, but it will depend on the graph and the particular initial triangulation T.

If we follow with the example of the network in Fig. 2.1 we can check that two different (minimal) triangulations can produce two different junction trees: those in fig. 2.4. Both JTs had added $\{F - B\}$ as fill-in in G_M^T, but in (a) the triangulation includes $\{A-E\}$ while in (b) it presents $\{B-D\}$ instead. And in this simple example, we also can notice a difference in the tree size, value that we seek to minimise. Notice that in terms of tree topology the same triangulation can also lead to distinct trees, although their total space size will be exactly the same, since they present the same set of clusters.

Figure 2.5 shows the construction (fusion of cliques in MPSs) for the join tree in Figure 2.4.a. In dotted lines we can see those cliques that can be fused. Notice that $\{F, E\}, \{B, E\}$ and $\{E, D\}$ are completely connected subgraphs in the moral graph

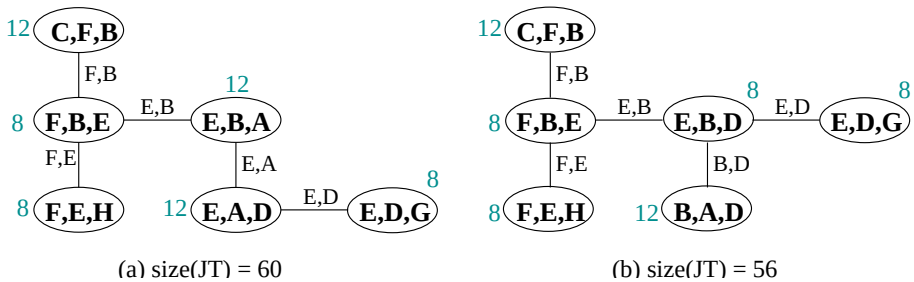


Figure 2.4: Two junction trees for the network in fig. 2.1.a. Next to each group/cluster its size is indicated.

of Figure 2.1, they subsequently did not need any merging to form a maximal prime subgraph. In this case, from JT 2.4.b we will obtain the same MPST. This does not always applies, since MPST structure depends on the initial tree (Fig. 2.6). Anyway, as we will see later the resulting MPS decomposition is effectively exactly the same for the same graph, because this is unique. The variations will only depend on the topology of the tree, producing one tree or another is not related to triangulation. Fill-ins are not considered in the completeness of a separator because only the moral graph is now regarded. Moreover, the example graph in fig. 2.6 is already triangulated (no need of fill-ins). Hence, here we observe that the distinction between two valid MPSTs comes from constructing a different JT and it is a consequence of choosing a father cluster from a set of possible ones (for example, in fig. 1.11(b)$[i = 3, v_i = X]$).

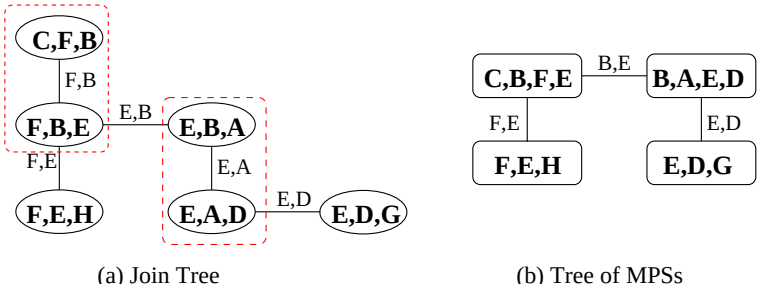


Figure 2.5: Construction of the MPSs tree and the obtained result.

We just finish this description enumerating and commenting the main properties/results shown for MPST:

1. The maximal prime subgraph decomposition of a moral graph G_M is unique.

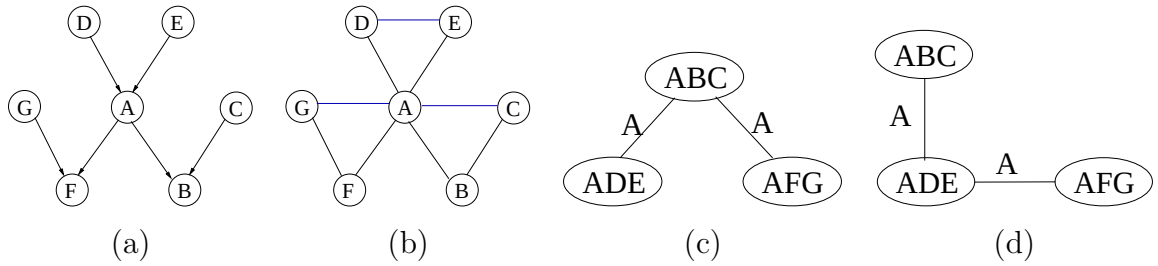


Figure 2.6: Another example of BN (a), where the moral graph (b) is already triangulated and at least two possible MPSTs exist, (c) and (d). Notice that here all cliques are MPSs, so JTs and MPTs coincide in both (c) and (d).

2. The clusters of $\mathcal{T}_{\mathcal{MPD}}$ formed by aggregation of cliques of a join tree $\mathcal{T}_{min}$ are not cliques in G_M.

3. The clusters of $\mathcal{T}_{\mathcal{MPD}}$ that are cliques of $\mathcal{T}_{min}$ are also cliques of G_M.

Result 3 (formulated as a theorem in [93]) yields the following corollary: "*If C is a clique of $\mathcal{T}_{min}$ and all separators connected to C are complete in G_M, then C is a clique of G_M.*".

These properties allow us to design the **ReTriangulation** method. The aim of triangulation is to produce a graph that is definitively decomposable in cliques. A maximal prime subgraph will always correspond to at least one clique or a set of them (aggregated). So it is an intermediate division between the graph and the junction tree that contains clusters (MPSs) with complete separators towards the rest of the graph. This fact prevents any node in the tree subgraphs from producing fill-ins connecting nodes not in this same subgraph[14]. We can also justify this point from the concept of MPS (def. 21) which is prime or irreducible, that is, two neighbour MPSs (in the MPS Tree) can be decomposed into two subgraphs by a clique separator and subsequently complete. However, for a single MPS is not possible to going further into decomposition. Suppose we have two subgraphs S_1 and S_2 adjacent in the tree. Then, $S_1 \cap S_2 = CS$ will always be complete for G_M and this guarantees that CS will also separate two cliques in any junction tree[15]: one belonging to the "aggregation" of cliques S_1 and the other one belonging to S_2 .

[14]This is the underlying idea of requiring a minimal triangulation for $\mathcal{T}$.

[15]That is why MPSD is done over the moral graph.

For all this, it is perfectly valid to triangulate every subgraph in an independent way from the rest, and make a global triangulation of the graph by the combination of these *partial* triangulations. And this will be the basis for the new *divide and conquer* triangulation process we describe next.

2.3.2 Triangulation of Bayesian networks by re-triangulation

Olesen and Madsen[93] identify a series of tasks that could benefit from the use of MPSD, among them BNs triangulation. The idea is to transform the BN triangulation into a set of triangulations, working separately/parallelly with each MPS.

Retriangulating a graph can be worthy, even when the same triangulation method is used twice. That is to say, the same triangulation method is applied (first) when triangulating the moral graph, and (secondly) when triangulating each MPS separately. As an example, let us consider the moral graph represented in Figure 2.1. As a preprocessing stage we could first remove (see the following paragraph) variables H and G (reduction). Then, using heuristic *minWeight*, the candidate variables to be removed in the first place are $\{A, C, D, F\}$, since all of them produce a clique of size 12. As ties are broken randomly, let us suppose D is the one selected, we then obtain the triangulated graph shown in Figure 2.1.c and the join tree shown in Figure 2.4.a (whose size is 60, if we assume all variables have two states). Now, if we construct the MPSD from this tree, we obtain the tree shown in Figure 2.5. If the subgraph $\{B, A, E, D\}$ is triangulated by using minWeight again, then the fill-in (B, D) will be added instead of the (E, A) and, therefore, the join tree of Figure 2.4.b will be obtained (whose size is 56).

The *reduction* preprocessing stage we have referred to in the previous example is a well known preprocessing rule [56, 9, 14] which consists of first deleting *simplicial* nodes (def. 17, that is, those nodes whose deletion does not introduce any fill-in, because they induce a completely connected subgraph in the moral graph). The application of this preprocessing rule decreases, sometimes drastically, the complexity of the triangulation task, because it starts from a smaller graph, and so the cardinality of the search space will be considerably smaller. In the moral graph depicted in Figure 2.1.b variables H and G are simplicial nodes.

The construction of the MPSD is not affected by the removal of simplicial nodes before the triangulation task begins, since all cliques obtained during this preprocessing

stage are in fact MPSs. This is because no fill-ins are added when a simplicial node is removed, so the separator set connecting it to the join tree is a complete subgraph in G_M.

Finally, the algorithm of ReTriangulation is as alg. 14 shows:

Algorithm 14 MPSD-based triangulation of a Bayesian network.

function ReTriangulation(BN)
▷ *We assume, as usual, that $BN = (G, P)$ and $G = (V, E)$.*
  $G_M \leftarrow$ Moralise(BN)
▷ *We could indicate G instead of BN, in this case the action of moralising is equivalent.*
  $G_M^R \leftarrow$ Reduce(G_M)
▷ *Obtain the reduced moral graph G_M^R. That is, remove simplicial nodes which are already triangulated.*
  $T_1 \leftarrow$ Get_Triangulation(G_M^R,*any_method*)
▷ *Triangulate G_M^R by using whichever triangulation method you prefer, since MPSD has been shown to be unique [93].* Get_Triangulation *returns a set of fill-ins, with a slight modification to alg. 7.*
  $T_1' \leftarrow$ Recursive_Thinning(T_1,G_M^R,T_1) ▷ *Now T_1' is minimal.*
  $JT \leftarrow$ Construct_Join_Tree(BN,T_1')
  $MPST \leftarrow$ Construct_MPS_Tree(JT,G_M) ▷ *Alg. 12.*
  MPSD $\mathcal{D} = \{S_1, \ldots, S_k\} \leftarrow$ Get_Decomposition($MPST$)
▷ *Notice this step is immediate, just given for clarity. $\mathcal{D}$ is then a decomposition of G_M^R.*
  for all $S_i \in \mathcal{D}$ **do**
    $T_i \leftarrow$ Get_Triangulation(G_i,*any_method*)
▷ *Triangulate each maximal prime subgraph S_i of $\mathcal{D}$. By* any_method *we mean that it could be used any triangulation technique. T_i will be the obtained triangulation.*
  end for
▷ *Notice that these triangulations can be done separately and in parallel.*
    return $T = \cup_{i=1}^{k} T_i$.
▷ *The union of all partial triangulations is returned. We could have applied recursive thinning again on T (optional).*
end function

2.3.2.1 Experimental Evaluation

In this section we present the experimentation carried out in order to study the impact of using the MPSD on the problem of triangulation.

- Used networks

  The test suite for the experimentation is made up of 9 complex networks taken from the repository[16] of the *Decision Support Systems Unit*[17] in Aalborg University, plus a tenth BN which is a subset of a *pedigree* network. In Table 2.1 we can see some relevant information about these networks: name, number of nodes and edges in both the moral and the reduced (without simplicial nodes) moral graph. We also show the number of MPS that come from the reduction process ($\#R$), and those obtained in the decomposition of the reduced moral graph ($\#D^R$), and the number of variables in the biggest MPS ($\#S^*$) of $\#D^R$. Finally, we also indicate information about the cliques in the reduced tree (T^R) obtained from G_M^R: the average number of cliques $\mu(\#cliques)$ and their average size $\mu(size(C_i))$. The total number of cliques in the obtained join tree for G_M is $\#cliques + \#R$.

Table 2.1: Some data about the BNs used in the experimentation.

	Moral Graph G_M		Reduced graph G_M^R		MPSs			T^R	
Network	$\#nodes$	$\#links$	$\#nodes$	$\#links$	$\#R$	$\#D^R$	$\#S^*$	$\mu(\#cliques)$	$\mu(size(C_i))$
Water	32	123	24	101	8	1	24	11	730079,182
Mildew	35	80	20	40	14	1	20	15	285122,667
Barley	48	126	35	92	12	2	33	24	822157,042
Munin1	189	366	108	241	69	1	108	93	2516647,896
Diabetes	413	819	335	665	76	1	335	261	74514,153
Pedigree4	441	806	163	305	223	4	155	144	15619,211
Link	724	1738	494	1349	218	1	494	373	2183041,696
Munin2	1003	1662	449	826	470	13	309	394	16024,985
Munin3	1044	1745	419	790	523	7	289	380	8453,979
Munin4	1041	1843	436	920	493	5	342	382	102551,711

Studying this table, we can observe that for some networks, all the nodes of G_M^R form a unique MPS. Clearly, in these cases the use of MPSD does not offer any advantage.

[16]The updated web page is: `http://www.cs.aau.dk/research/MI/Misc/networks.html`

[17]From 2005, *Machine Intelligence Group*.

- Triangulation methods used for our experiments

  As we have reviewed in section 2.2, there are many possible techniques to triangulate, and we have mainly divided them into two groups. Specific algorithms of both kinds have been proved in our experiments:

  1. *Greedy methods*. These are heuristic algorithms in which at each step we select the following variable to be deleted according to a determined criterion to be minimised, breaking ties randomly. The three most used criteria [63] are: the number of fill-ins to add when a variable X_i is deleted (*minFill*); the number of variables included in the clique formed with the elimination of a variable X_i (*minSize*); and the weight/size of the clique formed by eliminating a variable X_i (*minWeight*). With regard to the last criterion, in this study we will consider the heuristic proposed by Cano and Moral [18], which can be seen as an optimisation of *minWeight*. In this heuristic (*CanoMoral*), the size of the clique formed by the elimination of X_i is divided by $|\Omega_{X_i}|$(the number of different states for X_i), thus favouring the elimination of the variables with a greater number of states. The algorithms belonging to this approach provide a *good* solution quickly.

  2. *Genetic algorithms and other techniques of combinatorial optimisation* [71, 56, 64, 128, 47]. The idea is to apply an algorithm of metaheuristic nature taking as score function the state space size of the obtained join tree. Therefore, each individual or potential solution is represented as a deletion sequence and its score/fitness is calculated as the sum of the cliques' size obtained from the triangulation of G_M using this sequence. We try to minimise this score.

  In this study we will use the genetic algorithm (GA) proposed by Larrañaga et al.[71] with the modifications made in Gámez and Puerta [47]. Basically it is a *steady state* GA using the mutation and crossover operators that gave the best results in the experimentation carried out in Larrañaga et al. [71]. The modification made by Gámez and Puerta[47] consists of generating one half of the initial population randomly and the other half by means of the greedy algorithms previously mentioned, thereby helping to accelerate the convergence. On the other hand, the population size and the number of generations is calculated depending on the number of nodes of the graph [47].

  We should remark that these methods usually produce better results than the previous ones, but they require much more CPU time.

- Experiments design

  For each network with $\#D^R > 1$, we have carried out the following experiments:

  1. All the networks have been triangulated using greedy heuristic algorithms (minFill, minSize and CanoMoral) and genetic algorithms (GA). For the greedy algorithms each network has been triangulated 500 times. With respect to genetic algoritms, the number of runs is between 3 and 10 depending on the complexity of the network. Table 2.2 shows the results (on average) for the state space size $\mu(size_i)$ of the resultant join tree and the required CPU time $\mu(time_i)$ in seconds.

  2. As recursive thinning has to be applied before obtaining the MPS decomposition, we show in Table 2.3 the average CPU time for its application over each network $\mu(time_{ii})$. The effect of applying recursive thinning with respect to the state space size of the obtained join tree depends mainly on the triangulation technique. Thus, for the networks considered in our study this effect is quite important for the minFill criterion while for the rest (minSize, CanoMoral and GA) it has very little effect or none at all.

  3. Then, we obtain the MPS decomposition of each network, and each subgraph is (re)triangulated independently. The following retriangulations have been carried out:

  - Each subgraph is retriangulated using the same criterion (minFill, minSize, CanoMoral). In these three experiments 500 runs were carried out for each network.
  - Each subgraph is retriangulated using a genetic algorithm. In this experiment 10 runs were carried out for each network.
  - Finally, it seems that the success of each heuristic criterion considered in this study (minFill, minSize and CanoMoral) depends on the graph being triangulated. Therefore, the best choice in a MPS decomposition may be to apply different criteria to each subgraph. For this reason, we propose to use a combination of the three greedy heuristics. Thus, in this experiment each subgraph will be retriangulated three times, first with minFill, secondly with minSize and finally with CanoMoral. We then construct the triangulation of the whole graph by picking out the best solution found for each subgraph. This method will be notated as FSCM (because the three heuristic criteria used: minFill, minSize and CanoMoral). In this experiment 500 runs were carried out for each network.

The results obtained in these experiments are shown in table 2.4. In this table we show the average, the standard deviation and the best result obtained for state space size ($\mu(size_{ii})$, $sd(size_{ii})$, $best(size_{ii})$). The required CPU time (construction of the join tree graphical structure + MPSD + retriangulation) is also shown, $\mu(time_{iii})$.

Table 2.2: Results when triangulating the entire graph.

		minFill	minSize	CanoMoral	GA
Barley	$\mu(size_i)$	100.270.844,88	93.788.131,01	19.807.811,00	17.224.064,00
	$\mu(time_i)$	0,0060	0,0059	0,0059	17,1
Pedigree4	$\mu(size_i)$	1.135.222,88	2.218.223,23	2.199.486,96	574.781,40
	$\mu(time_i)$	0,226	0,2254	0,2254	5613,1
Munin2	$\mu(size_i)$	7.863.143,87	9.683.889,09	6.575.016,40	3.844.803,00
	$\mu(time_i)$	1,0192	1,0222	1,020	86765,8
Munin3	$\mu(size_i)$	7.980.516,15	3.349.001,45	3.290.505,65	3.099.098,67
	$\mu(time_i)$	1,1533	1,1485	1,1488	78626,2
Munin4	$\mu(size_i)$	36.931.253,19	31.685.429,42	39.118.690,00	12.836.469,67
	$\mu(time_i)$	1,1934	1,1942	1,1918	92260,0

Table 2.3: CPU time of applying *recursive thinning* to the previous triangulation.

	Barley	Pedigree4	Munin2	Munin3	Munin4
$\mu(time_{ii})$	0,0049	0,0521	0,4085	0,2834	0,4763

- Analysis of the experimental results

  From the study of the obtained results, we can state the following about the state space size of the resultant triangulation.

  – With respect to the comparison of applying the same greedy heuristics over the entire (reduced) graph and over each subgraph separately, it seems that on three of the five networks (Barley, Munin2 and Munin3) the quality of the obtained triangulation improved, while in the other two networks (Pedigree4 and Munin4) the differences are not so noticeable or indeed in certain cases the tree sizes are slightly worse.

Table 2.4: Results when triangulating each subgraph independently

Barley	$\mu(size_{ii})$	$sd(size_{ii})$	$best(size_{ii})$	$\mu(time_{iii})$
minFill	79.498.638,41	60.404.822,3	17.189.541	0,0239
minSize	80.768.310,89	74.433.727,8	17.247.576	0,0227
CanoMoral	19.807.811,00	0,0	19.807.811	0,0238
GA	17.261.540,00	180.603,6	17.140.796	14,3
FSCM	19.537.148,10	693.475,0	17.247.266	0,0648
Pedigree4	$\mu(size_{ii})$	$sd(size_{ii})$	$best(size_{ii})$	$\mu(time_{iii})$
minFill	1.112.433,70	410.181,6	612.873	0,5882
minSize	2.623.774,73	1.780.432,6	609.093	0,5745
CanoMoral	2.423.613,74	1.695.900,1	615.600	0,5771
GA	591.281,10	74.663,2	486.270	1331,9
FSCM	937.174,16	330.518,6	615.492	1,1864
Munin2	$\mu(size_{ii})$	$sd(size_{ii})$	$best(size_{ii})$	$\mu(time_{iii})$
minFill	4.648.878,25	1.435.180,4	2.490.948	12,1374
minSize	9.113.417,60	2.174.335,1	4.291.740	12,0303
CanoMoral	6.320.234,00	14.106,6	6.299.540	11,973
GA	3.511.745,70	485.903,4	2.781.769	12796,4
FSCM	3.677.987,29	633.450,7	2.371.904	15,2893
Munin3	$\mu(size_{ii})$	$sd(size_{ii})$	$best(size_{ii})$	$\mu(time_{iii})$
minFill	4.164.286,56	116.483,1	3.975.007	8,1726
minSize	3.292.094,68	20.142,1	3.263.225	8,0471
CanoMoral	3.258.919,65	15.716,2	3.246.682	7,9502
GA	3.102.859,20	27.168,6	3.077.688	10079,9
FSCM	3.250.990,61	3.789,1	3.242.927	10,763
Munin4	$\mu(size_{ii})$	$sd(size_{ii})$	$best(size_{ii})$	$\mu(time_{iii})$
minFill	23.638.148,95	1.988.804,0	19.696.135	10,4849
minSize	33.657.482,14	5.430.024,2	19.845.117	10,4098
CanoMoral	39.028.386,00	126.303,0	38.892.306	10,1736
GA	13.760.914,20	920.248,9	12.704.946	16811,4
FSCM	17.808.169,56	1.825.221,8	11.675.944	13,872

The discovery that we sometimes achieved (slightly) worse results when triangulating each subgraph separately as opposed to triangulating the entire graph, was quite a surprise for us. However, one explanation[18] could be that *although the maximal prime subgraphs can be triangulated optimally in an independent way in order to obtain an optimal result, the problem is caused by the heuristic nature of the triangulation algorithms. The triangulation algorithm is a zero step look-ahead algorithm. When triangulating the subgraph more degrees of freedom are available. Thus, due to the limited look-ahead the heuristic algorithm may choose to eliminate a node which will produce a bad result.*

- When genetic algorithms are used as the triangulation method, it seems that the results are quite similar with respect to these metrics (state space size). That is, when genetic algorithms are used for triangulating the entire graph they produce results similar to those achieved by the application of genetic algorithms to subgraphs, outperforming them in four of the five networks. This could be due to the *global* behaviour of genetic algorithms when performing the optimisation task.
- With respect to FSCM, it clearly overcomes the results obtained when just one greedy heuristic criterion is applied in the retriangulation. As we can see, FSCM sometimes yields join trees with a size two or four times smaller than that obtained by the individual application of minFill, minSize or CanoMoral.

  By applying FSCM we are taking advantage of the main feature of using MPS decomposition during the triangulation task, because it is in the nature of a network that a certain heuristic criterion perform better for only some of its portions, but in the other portions it might be wiser to choose a different criterion.

With respect to the efficiency of the algorithms, we can state the following about the required CPU time[19]:

- We have to bear in mind that the time values shown in Table 2.2 refer to the initial triangulation process. Since no graphical structure is necessary,

[18] This possible explanation was provided by an anonymous reviewer

[19] The experiments have been carried out using our own software written in Java, running over the virtual machine JDK 1.3 in a computer Pentium III 830 MHz and 512 MB of RAM

we only obtain the set of cliques. By contrast, the time values for the retriangulation (Table 2.4) show also the cost of building the join tree and the MPSD tree associated to it. In view of this, depending on the complexity of the network, the time for retriangulating is between 2 and 12 times greater than the initial triangulation time.

Taking into account the improvement in tree sizes, we feel that this result is quite good. In general, retriangulating gives better results, the most notable differences being found with the FSCM method. For example, for network Munin2 (449 nodes in G_M^R) it would take about 1.5 sec. to make the triangulation + RecursiveThinning, and about 15 seconds for JT construction + MPSD building + retriangulation.

- In the FSCM method the CPU time is greater than for the other heuristic methods, since each MPS is retriangulated 3 times. The difference is never more than 3 extra seconds, as the JT + MPSD construction is more CPU-consuming than the retriangulation process.
- Without doubt the greatest difference in regard to CPU time is seen in genetic algorithms, due to the number of generations and the population size computed according to the size of G_M^R. For example, for network Munin2, GA^s is almost 8 times faster than GA.

From the previous observations it is clear that genetic algorithms get better solutions than greedy heuristics, but they also need much more time. In our opinion, both kind of algorithms are really useful although at different stages of the probabilistic expert system development. In fact, we can distinguish (at least) two different situations in which the knowledge engineer performs the task of *compilation*:

- During the construction of the *knowledge base*, i.e., the Bayesian network. At this stage of the process, the knowledge engineer (KE) builds the network(s) in the *edit window* and tests the results of the modifications carried out in the *run window*[20]. Due to the fact that each time the KE switches from the edit window to the run window, a compilation occurs, and due also to the large

[20]This is the typical way of working in the most popular knowledge engineering tools oriented toward the development of PESs, as HUGIN (`www.hugin.dk`) or NETICA (`www.norsys.com`) or other research tools as Elvira (`http://leo.ugr.es/elvira/`) and GeNIe (`http://www.sis.pitt.edu/~genie/`).

number of modifications carried out by the KE over the network before the final version is obtained, it is obvious there is a need for a fast compilation process. Deterministic heuristics are therefore particularly suitable at this stage. We will refer to compilation at this stage as *on-line* compilation.

– Before the final product is given to the user. At this stage, our goal should be to produce a join tree as good as possible, because hundreds or thousands of propagations will be carried out over it. At this stage we can therefore spend more time on the compilation process, in order to achieve a better join tree. Algorithms requiring more CPU time are also suitable. We will refer to compilation at this stage as *off-line* compilation.

  As an example let us consider the Pedigree4 network. For this network, propagation over a join tree of size about 1.15e+6 requires about 800 seconds, while propagation over a join tree of size about 5.7e+5 requires less than 300 seconds, by using the platform described in footnote 20. In view of these values, it is worthwhile producing a very good tree, if time permits.

- Discussion about the method

  With this work, we have tested our method and studied the applicability of the MPSD to the problem of triangulation in BNs. At first blush, this technique seems to be quite promising because of the great number of MPSs in which the moral graph is usually decomposed. However, a more detailed study shows us that many times most of these subgraphs come directly from the reduction process, so that they do not need to be triangulated. If we exclude these subgraphs from our analysis, in many cases (half of the networks studied here) only one MPS is left, so that we are in the position as with general triangulation. In the other cases it is usual to have one MPS that is much bigger than the rest[93]. Nevertheless, as we have observed in the experimentation, in these cases the retriangulation is normally profitable with regard to the obtained result (specially when applying the FSCM algorithm), or with regard to CPU time in case of a genetic algorithm.

  In conclusion, our aim is to show that since the MPSD can be rapidly obtained from a join tree, this procedure is well worth carrying out, the decision whether or not to apply retriangulation being taken on the basis of the number of obtained MPSs and their sizes.

2.4 Main conclusions and further research.

From the previous point we can mainly conclude that there exist some possibilities to optimise these results and to explore new combinations to get even better triangulations.

We think that it may be feasible to design an α-MPS triangulation method (α-completeness will be roughly introduced at the end of chapter 4). Maximal Prime Subgraphs present really interesting properties based on graph theory results. The drawback is that for some real cases this decomposition given by MPSs turns out to be a little unbalanced. Therefore, for that networks the benefit of MPSD is reduced. So, we are working on some kind of heuristic/approximate decomposition method which, combined with MPSD, could produce better results for those particular cases.

Another idea that can be tested in order to study the behaviour of the ReTriangulation technique is the use of LB-triang method when obtaining initial elimination orderings, since this produces minimal triangulations and is claimed to be very efficient.

Finally, since MPSs can be triangulated independently among them, it seems a nice try to test how a real parallel execution could impact on the performance of the triangulation process.

On the other hand, from this whole chapter and the previously analysed issues we can draw the following main conclusions:

- Triangulation is still an unsolved problem, at least for a general case.
- But triangulating has also been proved to be an unavoidable step in the computation of Bayesian networks.
- This necessity for triangulating has brought about several endeavours to handle this problem, and the techniques found in literature are of distinct nature. In our case we tried to exploit the natural decomposition of a graph into its prime subgraphs.
- In any manner, this decomposition tool is not reserved for triangulation itself, it can become even more powerful. The use of MPSD can be extended to the whole process of compilation. Since triangulation is the most *expensive* phase of compilation and this can be correctly and separately distributed among MPSs, we could sketch other techniques so that compilation could be less "dependent" on the global triangulation. For that, we propose to look more closely into the

possibility of retriangulating some portions of the BN, and to use MPSD to perform incremental triangulations. This idea leads us directly to next chapter that will expound our approach for Incremental Compilation of Bayesian networks.

Chapter 3

Incremental compilation of Bayesian networks

Technology is the effort to spare an effort.
José Ortega y Gasset. (1883–1955)
Spanish philosopher and essayist.

3.1 The interest of designing an IC method

3.1.1 Motivation

When a Bayesian network is modified, for example adding or deleting a node, or changing the probability distributions, we usually will need a total recompilation of the model, despite *feeling* that a partial (re)compilation could have been enough.

Specially when considering dynamic models, in which variables are added and removed very frequently, these recompilations are quite resource consuming. But even further, for the task of building a model, which is in many occasions an iterative process, there is a clear lack of flexibility.

When we use the term *Incremental Compilation* or IC we just refer to the possibility of modifying a network and avoiding a complete recompilation to obtain the new (and different) join tree.

In the previous chapters we have been looking into the compilation process, which takes a considerable amount of time to be completed, particularly for big networks. Hence, incremental compilation tries to give an answer to the following question: "*When modifying a Bayesian network, is it absolutely necessary to recompile it from scratch?*". We could guess that if this modification does not affect the network globally,

it might be possible to find a manner to save some time in obtaining the associated tree. Fig. 3.1 tries to illustrate this simple idea: let us assume that we have an initial Bayesian network BN and its associated join tree $\mathcal{T}$, obtained by a compilation process. Later, at a certain moment, a set of modifications have to be done on this initial network, which will be transformed into a different network BN'. For this new network $\mathcal{T}$ will clearly not be a valid join tree, but it must be another one $\mathcal{T}'$ which is. The key point for the incremental compilation would then be to find a way (obviously, quicker than a complete recompilation of BN') that allows us to reach from this new network the/a new tree that corresponds to it. Thus, somehow, the question we previously launched could be reformulated as follows: "*If we know the performed modifications in our network, and from the initial join tree corresponding to the initial network ($\mathcal{T}$), could we reach the new join tree ($\mathcal{T}'$)?*". This is precisely the question annotated as a question mark in fig. 3.1, since this is exactly what we are looking for. And the answer will be our concrete method to carry out this incremental compilation process as section 3.2 describes.

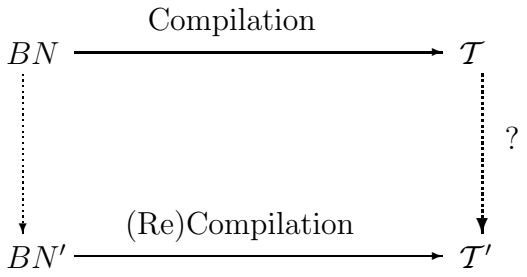


Figure 3.1: Scheme that represents the general idea for our approach for the incremental compilation of a Bayesian network BN.

3.1.2 Previous works

Little attention has been directed towards incremental compilation in the literature. Darwiche considers dynamic generation of join trees [27], but his aim is to produce efficient elimination orders for specific queries posed to the model rather than modifications over the network. His method is focused on efficient processing by online generation of specific join trees and it does not take modifications of the structure of the underlying network into account. There is another work by Draper [38] that will be reviewed later in which the main goal is to show how to build join trees avoiding

(thinking about) triangulation. To do this, a family graph is initially computed and then transformed into a join tree by using a set of heuristics. This approach can also be applied in case of dynamic changes in the network structure where the same set of heuristics are applied to dynamically maintain the join tree. A disadvantage of the method is that it is difficult to identify the relevant set of heuristics and that the resulting join trees are often suboptimal.

Thus, this issue is relatively new in the framework of Bayesian networks, and has been little studied in the literature. In 1999, Olesen and Madsen [93] launched the idea of using MPSD to perform an IC. I considered this idea in 2000 for my MSc degree report [41], where the problem was approached in a systematic way. Like that, in this work, it was shown that for simple modifications, the complete construction of the new tree was unnecessary. Single modifications were used, that is, in this work there is a study of the impact on the join tree $\mathcal{T}'$ in the moment of adding or removing a single element (node/variable or arc/link) and also, when modifying the table of associated probabilities.

The cases that were included in this preliminary work are:

1. **Potentials**: changing the probability values (the simplest case).

2. **Graph**: changing the network structure and, then, the associated moral graph.

   (a) Variables or nodes: varying the information about nodes in the graph, in such a way that

      i. We could modify the number of **states** that the variable has. For example, it could happen that at a certain moment we find suitable to add a new case in the set of possible states for a particular variable

      ii. Or we could also realise that a certain variable does not introduce any new information for the system and we decide its **elimination**.

      iii. Finally, when modifying a network, it might arrive that we find interesting to consider a new factor that we have not previously foreseen, which will directly take us to the **insertion** of a new node.

   (b) Links: similarly to the variables case, we could introduce changes in a network with respect to its arcs.

      i. We could detect that a relation does not really reflect on the system, producing then the **elimination** of the corresponding link.

ii. To the contrary, it is possible that we identify a direct relation/dependency between two variables that we did not previously catch on the network. Hence, this will yield the **insertion** of a new link.

Apart from this classification of the single modifications, in the work [41] it was observed that we should take into account which node or which link we are modifying. Because of that, in this first try of incremental compilation, it was distinguished, for example, among nodes without parents, nodes that do not have children, nodes that are parents of one unique node, nodes with various children,... And in an analogous way, the links were treated differently according to their nature, if they were involved in an undirected cycle, if they had caused a moral link or a fill-in edge,...

Easy examples for all these cases were studying using the Asia network, a simple one, but still presenting some of the most representative cases in a Bayesian network. In [41] an example for each of the previously listed cases is found, the particular modification is studied and we proposed an alternative solution to the complete recompilation. In a reasoned way, but not fully theoretically demonstrated, this situation was extended to all similar (single) cases provided that certain requirements hold. Also, a complete recompilation is performed in order to compare the two resulting trees.

In some cases the proposed solution gave rise to an identical tree to that reached after a total recompilation. In other cases, the obtained tree was valid but of less quality (larger total state space size) than the one obtained by the traditional process. And finally, in the most complex cases[1] a clear solution for the problem was not found.

3.1.3 Decomposition of the problem

So, in the commented work, there were some cases that could not be solved and for those that could, the given alternative solution resulted too specific, which took us to think of the problem from a different perspective: using the graph decomposition (MPSD in our case) in subgroups that could be treated independently. This decomposition has been studied for independent triangulations (chapter 2), but it also seems valid for reusing old triangulations that do not need to be reprocessed. This technique will be thoroughly explained along all the remaining of this chapter.

Before going directly to our designed method, we should first make a comment on a different decomposition method published in 1995 [38]. This paper was presented

[1]Even though modifications were single, when a node was involved in too many families it was difficult to find a general solution.

as an alternative way to tackle the clustering task that underlies the notion of join tree. They take a graph (not a tree) of clusters that must fulfill a modified version of the running intersection property of the join trees. And this graph is posteriorly and iteratively transformed until finally finding a tree. That is, the main goal of this work is to skip the triangulation step as it is normally performed, and work with the modifications directly on the graph of clusters. They present distinct methods to reflect in this graph the modifications done in the network. They start from an empty network and the technique proceeds to add its elements incrementally, obtaining after each modification the new join tree. The results are quite satisfying, but this method presents some main drawbacks:

- Although its main foundation is to reach the join tree without thinking in triangulation, the triangulation process is implicitly in the transformations the graph is undergoing.
- The processing that is carried out implies a great number of verifications, and the fact that this is performed from the empty network until the complete one step by step makes its execution very slow.
- Finally, and even more important, the obtained trees, are in most of the cases non-optimal, and they turn out to be quite worse than those reached by means of a traditional triangulation using any of the heuristic methods reviewed in the previous chapter.

Anyway Draper's work resulted quite enriching to understand and to gain an insight to start a deeper study of the possibility for an incremental compilation.

The feasibility of reusing an already existing JT in order to obtain the new one regarding only the modifications in the network had not really been studied thoroughly before. In this chapter we present a method for incremental compilation of a Bayesian network, following the classical scheme in which triangulation plays the key role. In order to perform incremental compilation we propose to recompile only those parts of the JT which may have been affected by the network's modifications. To do so, we exploit the technique of maximal prime subgraph decomposition in determining the minimal set of subgraphs that have to be recompiled, and thereby the minimal subtree(s) of the JT that should be replaced by new subtree(s). Changes are normally located in the same area, and if the network is large, a set of changes will typically influence only a little part of the network. These two reasons encouraged us to work on a compilation that could be carried out partially.

3.1.4 Introductory issues

Most popular knowledge engineering tools for construction and execution of probabilistic expert systems based on Bayesian networks (BNs), such as for example HUGIN [59] or NETICA [89], work with two representations: 1.- A direct representation of the BN as illustrated by the edit window in figure 3.2 and 2.- A computational structure also known as the junction tree or the join tree (JT), illustrated by the run window in figure 3.2.

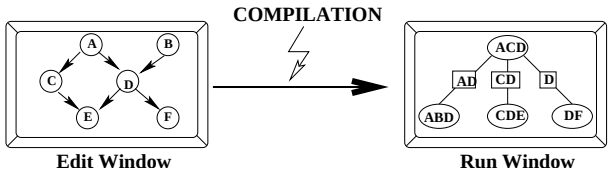


Figure 3.2: Two representations of a Bayesian network.

A BN is constructed or modified in the first representation and inference is performed over the second representation. Each time the knowledge engineer switches from editing to execution, a *compilation* builds the join tree from scratch. If the network is large, then the CPU time required to perform a new compilation will be considerable. Therefore, it will be desirable that once a join tree representation of the Bayesian network has been generated, incremental changes in the network should produce an update of the previous join tree, and not a new full compilation process. Apart from the *efficiency* reason, there could be more reasons to prefer incremental compilation over total compilation. Draper emphasises *stability* of the join tree as a desirable property of incremental compilation [38]. It is anticipated that a considerable effort is made to produce an efficient join tree and that incremental changes to an existing (near) optimal join tree will produce more stable results.

The method we propose for incremental compilation identifies the parts of the join tree that are affected by changes in the BN, reconstructs only those parts of the join tree and glues the new substructures into the original join tree instead of the outdated parts. This approach ensures a stable and, most of the times, efficient resulting join tree. The method builds on a third representation of the underlying BN the *Maximal Prime subgraph Decomposition (MPD) tree* [93]. As seen in chapter 2, maximal prime subgraph is a subgraph that is d-separated from its surroundings by complete (i.e. fully connected) separators, which enables divide and conquer algorithms for various graph operations, triangulations for instance. There is a direct correspondence between

maximal prime subgraphs (MPS) and subtrees in the join tree, such that a maximal prime subgraph always corresponds to one or more cliques in the join tree. It is this correspondence that is exploited in the present work, by identification of the MPSs that are affected by modifications in the BN. These MPSs determine the parts of the join tree that have to be altered, and an updated join tree can then be constructed incrementally.

3.2 MPSD-Based Incremental Compilation

3.2.1 The role of MPSD within Incremental Compilation

Our main concern is therefore to get a general procedure to solve the problem of getting to the new join tree $\mathcal{T}'$ from the initial one $\mathcal{T}$ (plus the performed changes) as Fig. 3.1 illustrates. And we have already anticipated our proposal is an algorithm that utilises the MPSD. The Maximal Prime Subgraph Tree ($\mathcal{T}_{MPD}$) is, as explained in chapter 2, a structure obtainable from the join tree. To remember the necessary steps to attain from one to the other we have summarised them in figure 3.3. Starting from the Bayesian network BN, one of its components is the graph G (the other one is the probability distribution), from which we compute the moral graph G^m. Immediately afterward(s), the graph is triangulated, being this triangulation necessarily minimal, which gives rise to $G^{T_{min}}$. Last, from this chordal graph we can construct the corresponding join tree, $\mathcal{T}$ and with algorithms 11 and 12 we can identify the maximal prime subgraphs and get the tree $\mathcal{T}_{\mathcal{MPD}}$.

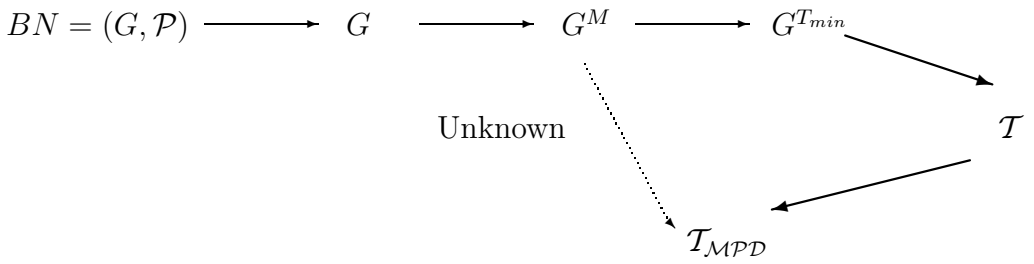


Figure 3.3: Graphical process that indicates how to reach the MPST $\mathcal{T}_{\mathcal{MPD}}$ from a Bayesian network BN, using as an intermediate step the join tree $\mathcal{T}$.

What is the role of the MPS Tree in our alternative compilation process? As fig. 3.1 represents, we were searching a manner to go from $\mathcal{T}$ to the new tree $\mathcal{T}'$, without requiring a full recompilation. And the MPS Tree $\mathcal{T}_{\mathcal{MPD}}$ can be understood as an intermediate structure between the BN and the tree $\mathcal{T}$, bearing also in mind that (see figure 3.3) from this one ($\mathcal{T}$) $\mathcal{T}_{\mathcal{MPD}}$ can be easily construted.

Hence, the intention now is to go through the (previously questioned) path from the original join tree to the modified one passing by their corresponding MPSTs. Figure 3.4 attempts to illustrate this path. Question marks refer to the two steps for which the execution is not initially clear, while the rest has just been described (see fig. 3.3 or in detail chapter 2). It is precisely these two particular unknown points that our proposed algorithm will give an answer for.

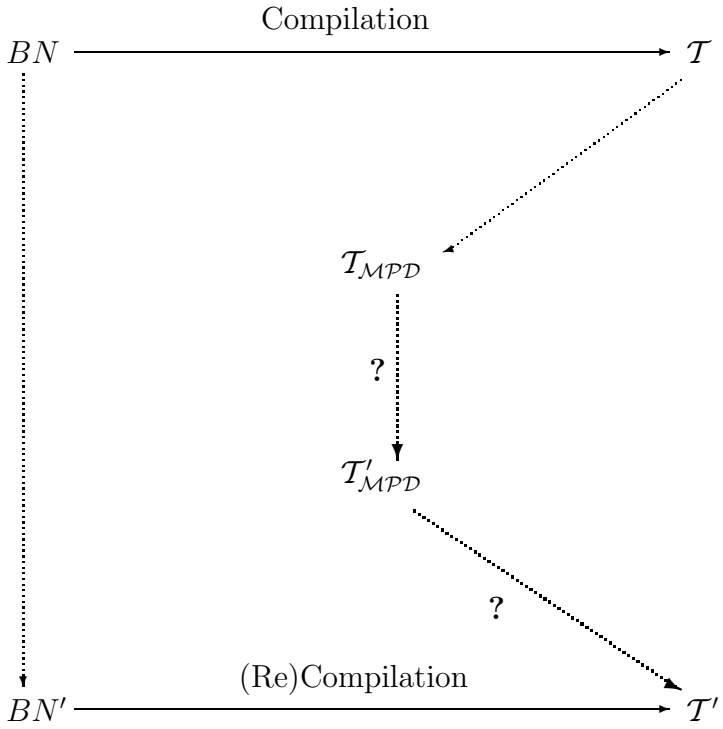


Figure 3.4: Sketch that represents an overview of our Incremental Compilation process for a Bayesian network BN by means of the MPSD.

3.2.2 Possible modifications to be considered

The starting point in our approach to incremental compilation is a set of changes in an existing BN. Such changes range from simple modifications, e.g. adjustments

of numerical parameters in the (conditional) distributions for variables, to complex structural reorganisation of variables and their links, possibly altering large parts of the BN. In the following we will briefly go through the possible modifications we will look into. They will be close to the set of variations considered in the preliminary approach in [41]. Nevertheless, there are two outstanding novelties: first, this view is done towards the generation of a general (and more efficient), always valid method that could process all kinds of situations; and second, we also wish a method able of treating more than one modification at a time, that is, able of processing group of changes as well as single ones, according to user preferences.

3.2.2.1 Modification of potentials

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

3.2.2.2 Modification of the states of a variable

Changes in the state space of a variable can alter the structure of the optimal join tree, because the triangulation typically takes the state space of variables into account. Notice that these state variations will not affect when treewidth is considered. A full treatment of incremental compilation should, of course, consider this class of changes, but the overall efficiency of the resulting computational structure remains roughly the same. We shall therefore disregard the structural implications of such changes from further considerations in the present treatment of the subject. What remains is then the modified structure of the potentials of the altered variable and its children and this is again taken care of by replacing old potentials with new ones for the affected variables.

3.2.2.3 Modifying the graph structure

- Removing an arc

Removal of an arc in a BN can be a straightforward change. If, for example, two nodes without parents share a child and is not otherwise connected, removal of an arc between them will not lead to changes in the moral graph, as the link would appear

here anyway due to moralisation. At the other extreme the removal of an arc could break several loops in the BN and a considerable simpler join tree could result from a retriangulation of the network. In such cases it is beneficial to be able to identify a minimal part of the join tree that could be affected by the change and concentrate on a retriangulation of only that part.

•Adding an arc

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

•Removing a node

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

•Adding a node

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

3.2.3 MPS as a tool for Incremental Compilation

In this chapter we are primarily interested in structural changes over the network. We shall therefore concentrate on the addition and deletion of nodes and arcs. The problem we investigate is summarised in figure 3.1, and partially answered in 3.4. As mentioned earlier, modifications of the BN can result in everything from trivial to very

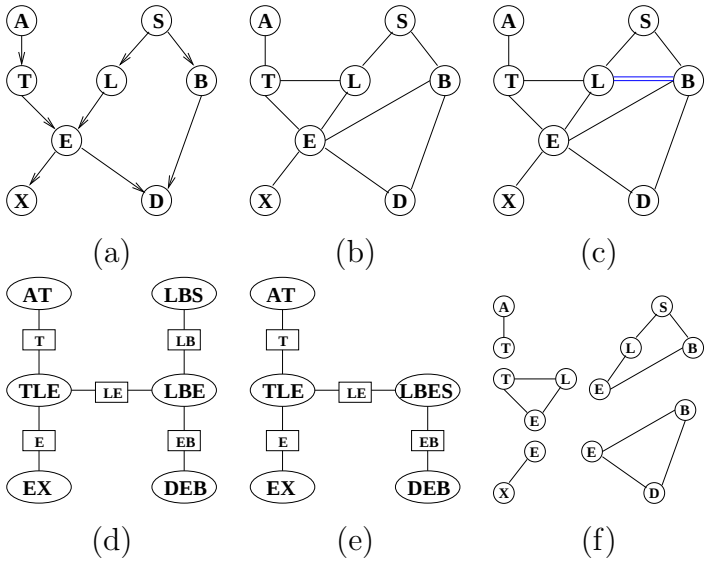


Figure 3.5: (a) The Asia network. (b) Moral graph for Asia. (c) Triangulated graph for Asia. (d) A join tree for Asia. (e) Maximal prime subgraph tree for Asia. (f) Maximal prime subgraph decomposition for Asia.

complex modifications of the join tree. We are therefore looking for ways to limit the parts of the join tree that have to be recompiled. The maximal prime subgraphs of the graph of the BN is the intermediate structure that can resolve this issue. We should therefore review the method by Olesen and Madsen [93] for identification of a maximal prime subgraph decomposition explained in previous chapter.

The decomposition of the graph of a Bayesian network into its maximal prime subgraphs is integrated into the well known procedure for construction of join trees for Bayesian networks. We know that the maximal prime subgraphs of G^M are formed by aggregating adjacent cliques of $\mathcal{T}_{min}$ connected by a separator which is incomplete in G^M. Algorithm CONSTRUCT_MPD_TREE (alg. 12) performs this process and returns a join tree $\mathcal{T}_{MPD}$, where the nodes represents the maximal prime subgraphs.

Figure 3.5 illustrates the application of the algorithms on the well-known Asia example. Part (a) shows the BN for the example and in part (b) the moral graph is obtained by adding arcs between common parents of all nodes (arcs (T, L) and (E, B)), and dropping the directions of the original arcs. In part (c) the triangulated graph results from adding the arc (L, B). The triangulation is minimal and no arcs have to be removed by the recursive thinning step. The resulting join tree is shown in part (d) and a check of the separators in the moral graph yields the maximal prime subgraph decomposition tree shown in part (e). Part (f) gives the maximal prime subgraphs of

the Asia network. Let us remember that this decomposition is unique.

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

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

Before explaining in details how to cope with the four basic modifications (remove arc, add arc, remove variable and add variable), we give the general procedure to perform *MPSD-based incremental compilation*. Figure 3.6 illustrates the whole process and will be referred during the explanation provided below. Notice that this figure summarises those given before to reason the way this method was being designed and developed.

Let us suppose that the process starts with a user making modifications in the edit window of figure 3.2. The result of this editing process is an updated version of the BN (G') (step (1) in figure 3.6). A list of the modifications[2] is constructed during this step. This list serves as input to algorithm 3, that is activated when the user decides to obtain a new join tree.

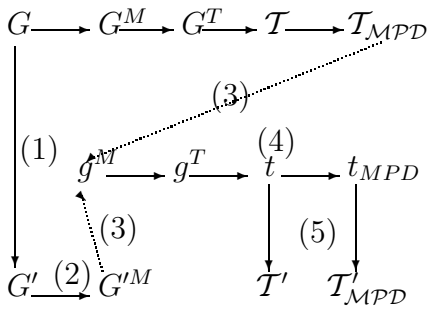


Figure 3.6: Integrated overview of the MPSD-based incremental compilation process.

[2]Remark how we can group modifications as we want.

Once the user decides to produce a join tree for the new network G', the first step of incremental compilation is to modify the moral graph (Step (2) in figure 3.6). Notice that the moral graph plays a crucial role in MPSD and $\mathcal{T}_{MPD}$ construction. We will pay special attention to this step in algorithm 16.

As an example, let us suppose that the user has removed link $L \rightarrow E$ and has invoked the IC algorithm. Then, the updated moral graph is the one depicted in part (b) (and the modified BN is shown in part (a) of figure 3.7). Remark that the moral link T-L has also been removed.

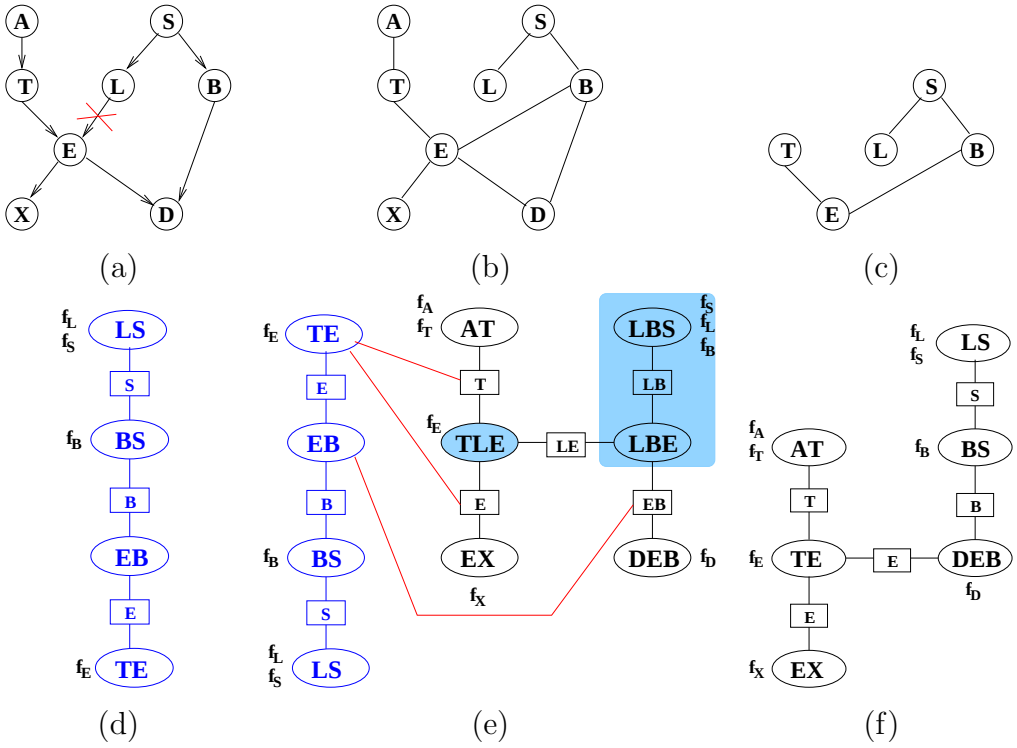


Figure 3.7: MPSD-based incremental compilation of Asia when removing $L \rightarrow E$.

Now, we proceed to step (3) in figure 3.6. This is the key point in our algorithm, where we identify the minimal set of MPSs which may be affected by the modifications performed over the BN. Shortly we shall detail this for the different modifications, but for the moment, let us assume the existence of an algorithm which marks the MPSs in $\mathcal{T}_{MPD}$ affected by a modification. In the example, this algorithm marks MPSs (TLE) and $(LBES)$ (to be detailed in subsection 4.1). In the example, there is only one connected marked subtree $(TLE) - [LE] - (LBES)$, but in general, when all modifications have been processed by the marking algorithm, there may be more connected parts of $\mathcal{T}_{MPD}$ that have been marked. As an example, consider the scenario

in which the marking algorithm marks MPSs: (AT), (TLE) and (DEB); which gives rise to two marked connected subtrees: $(AT) - [T] - (TLE)$ and (DEB). For each of these marked connected subtrees we proceed, in turn, with the following steps.

Let g^M be the subgraph of G'^M induced by the set of variables included in a connected marked subtree of $\mathcal{T}_{MPD}$. Figure 3.7.c shows g^M as the projection of G^M over $\{T, L, E, B, S\}$. In step (4) in figure 3.6 we obtain a JT and a MPD tree for g^M. We obtain a join tree t by applying algorithm 11 (avoiding step in line 1, since the graph is already moralised) and the corresponding MPD tree t_{MPD} by applying algorithm 11. In the example, part (d) of figure 3.7 shows both structures, because in this case every MPS corresponds uniquely to one clique.

Finally, during step (5) in figure 3.6 both $\mathcal{T}$ and $\mathcal{T}_{MPD}$ are updated by using the newly obtained structures t and t_{MPD}. The process is completely analogous in both cases and it only differs in the tree to which it is applied.For each separator S connecting a marked subtree with an unmarked cluster we reconnect S to a cluster of t (t_{MPD}) having maximal intersection with S. Upon completion, the marked clusters (and separators between them) are deleted. In the example, the separators connecting the marked connected subtree with the rest of $\mathcal{T}_{MPD}$ are $[T], [E]$ and $[EB]$. These separators are connected with the new graphical structure as shown in part (e) of figure 3.7. Finally, part (f) of the same figure shows the final structure, obtained after removing the outdated part of the tree (the marked MPSs). Algorithm 17 details this procedure.

3.2.4 Incremental Compilation Algorithm

Algorithm 15 details the incremental compilation process described above. In order to simplify the header of the algorithms presented in this section, we suppose that the main graphical structures are accessible, that is, we will refer to G^M, $\mathcal{T}$ and $\mathcal{T}_{MPD}$ without the need of passing them as a parameter to the algorithms. Before we proceed we need some notation. The potential of X is assigned to a specific clique in $\mathcal{T}$, which contains the family of X. In the following, we will use C_X to identify this clique and, likewise, M_X will identify the MPS in $\mathcal{T}_{MPD}$ which has the family of X associated. In figures this will be indicated with a f_X next to the corresponding clique/MPS.

The first loop of algorithm 15 iterates over all modifications. For each modification we adjust the moral graph by algorithm 16. Algorithm 16 maintains a list of all links that are affected by the actual modification. This is relevant for addition and deletion of arcs, and algorithm 16 returns a list, L, with the added (deleted) link and with the

induced added (deleted) moral links.

Algorithm 15 Performs Incremental Compilation when introducing a list of modifications in the network.

```
1: procedure IncrementalCompilation (Modification list ModList)
2:     for all Modificaction mod ∈ ModList do
3:         L ← ModifyMoralGraph(mod)
4:         switch mod do
5:             Case Add node X: AddNode(X)
6:             Case Delete node X: RemoveNode(X,M_X,nil)
7:             Case Delete link X → Y: RemoveLink(L,M_Y,nil)
8:             Case Add link X → Y: AddLink(L)
9:         end switch
10:    end for
11:    for all Connected marked subtree T_MPD ∈ 𝒯_MPD do
12:        T ← 𝒯 corresponding to T_MPD
13:        for all Clique C_i ∈ T do MarkClique(C_i)
14:        end for
15:        C ← any cluster of T
16:        M ← any cluster of T_MPD
17:        V ← {all variables included in T_MPD}
18:        t ← ConstructJoinTree(g^M)
   ▷ g^M is the projection of the current moral graph G^M over the set of variables included in T_MPD.
19:        t_MPD ← AggregateCliques(t)
20:        𝒯 ← connect(t, C, nil)
21:        𝒯_MPD ← connect(t_MPD, M, nil)
22:        delete(T)
23:        delete(T_MPD)
24:    end for
25: end procedure
```

Algorithm 16 Performs the corresponding modifications to the moral graph implied by the *mod* which is being processed.

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

The list, L, returned by algorithm 16 is passed on as argument to the relevant procedure, that marks affected MPDs in $\mathcal{T}_{MPD}$. This result of steps 1-10 in algorithm 15 is an MPD-tree with (possible several) connected marked subtrees. In the second part of algorithm 15 (lines 11-24) we iterate over these subtrees and adjusts $\mathcal{T}$ and $\mathcal{T}_{MPD}$ by algorithm 17. The pattern for this algorithm may not be immediately transparent. The recursive control structure acts on the two last parameters where the former (the second parameter) is the cluster to which the algorithm is applied, and the latter (the third parameter) is the caller. The structure traverses a marked subtree, avoiding loops by a check that the caller is not re-visited. This pattern is also used in algorithms 18 and 19.

Algorithm 17 Performs the connection between the new *partial* (JT or MPS) tree to the old remaining part.

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

We shall now go through the details of the marking of MPDs, that is, the procedures called from the first loop of algorithm 15. It is silently assumed that whenever a MPD is marked, the corresponding cliques in the join tree will also be marked.

3.2.5 Removing a link

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

The following algorithm marks the MPSs affected by the removal of $X \rightarrow Y$. Parameter L is the list of (induced) moral links (returned by ModifyMoralGraph). Notice, that when the algorithm is called the first time (from Incremental Compilation) then $M_Z = nil$.

As a case of study, let us pick up the example used during the overview of our IC method, that is, the removing of link $L \rightarrow E$ from the Asia network. In this case, the parameters received by algorithm 18 are $L = \{(L, E), (T, L)\}$, $M_Y = (TLE)$ and $M_Z = nil$. Therefore, MPS (TLE) is marked in step 2. From the three separators

connected to (TLE), only $[LE]$ will be considered, because the other two contain only one variable. As $[LE]$ contains one of the removed links, the MPS $(LBES)$ connected to (TLE) by this separator, is also marked by a recursive call of RemoveLink. Therefore, in this case, the subtree $(TLE) - [LE] - (LBES)$ is marked by the algorithm.

Algorithm 18 Algorithm that performs the marking of subgraphs when deleting a link in the Incremental Compilation process.

```
1: procedure RemoveLink(LinkList L,MPS M_Y,MPS M_Z)
2:     Mark M_Y
3:     for all Neighbour M_K ≠ M_Z of M_Y do
4:         S ← separator between M_Y and M_K
5:         if L ∩ links(S) ≠ ∅ then
6:             RemoveLink(L, M_K, M_Y)
7:         end if
8:     end for
9: end procedure
```

One could think that the recursive call of RemoveLink is avoidable. Effectively, in some cases this will not be necessary, but in some other occasions this further checking is absolutely needed to produce the valid result. To prove that, we are going to present a simple example. In fig. 3.8.(a) the BN is depicted whereas fig. 3.8.(b) represents the corresponding moral graph. Notice that this moral graph is already triangulated, to check that it is enough to use, for example, deletion sequence $\sigma = \{I, H, D, A, F, C, G, B\}$. This elimination order will not introduce any fill-in, and the initial graph is then already triangulated ($T_{min} = \emptyset$).

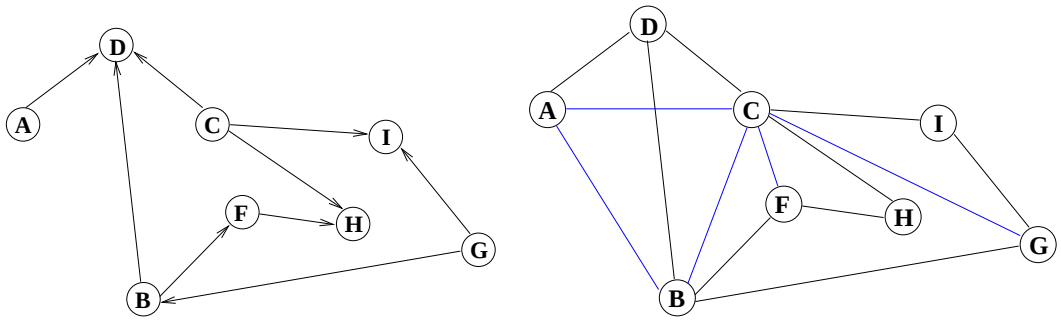


Figure 3.8: Network example to prove the necessity of recursively RemoveLink.

The corresponding tree will be the one in figure 3.9. Notice that families for C

and G could have been located in another clique/mps containing this variable. Let us suppose that we intend to remove the link $B \rightarrow D$. The MPS which contains the family of D, M_D is [ABCD]. The elimination of this arc also makes the moral links $B - C$ and $A - C$ disappear (as illustrated in fig. 3.10.(b)).

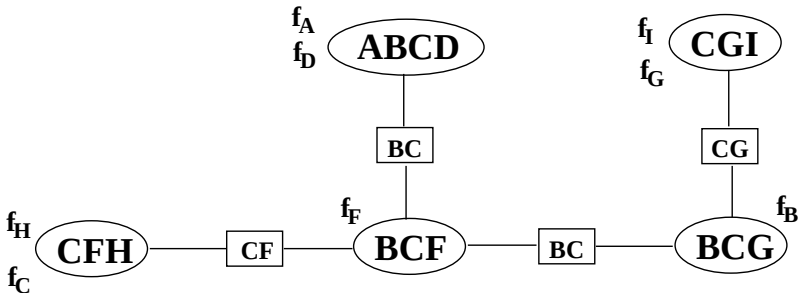


Figure 3.9: Join Tree and MPD Tree (they are equivalent) for network in figure 3.8.

In the removing processing of this link, as $B - C$ is in L we should call then RemoveLink(L,[BCF],[ABCD]) and these two MPSs will be marked. The question now would be if we stop there or we need to go further. In case that we stop, the IC process is summarised in figure 3.11. It is clear (see fig. 3.11.(c)) that this does not produce a valid tree. The marked part is the retriangulated portion and disappears because it is substituted by the new t_{MPD}. Nevertheless, this connection task gives rise to an erroneous situation: if $[BF]$ is joined to the $[CFH]$ node ($[BF] \cap [CFH] = F$) it will not be a tree anymore, since there is a cycle and besides the tree would become inexplicably disconnected. Otherwise, if $[BF]$ is joined to the $[BCG]$ node ($[BF] \cap [BCG] = B$) the running intersection property is violated, since there would be a path from $[BCG]$ to $[CFH]$ where their intersection, C, is not contained.

In this example, if we had followed with the recursive call RemoveLink(L,[BCG], [BCF]) the MPS $[BCG]$ would have been also marked and the process will be as depicted in figure 3.12. Notice that g^M has been triangulated adding the fill-in $F - G$ and that is why separator $[FG]$ is not complete in the moral graph, and it therefore aggregates two cliques in one MPS of t_{MPD}. Remark that now the process can be executed smoothly and that the resulting tree is the correct one (equal to the one in fig. 3.10.(d)).

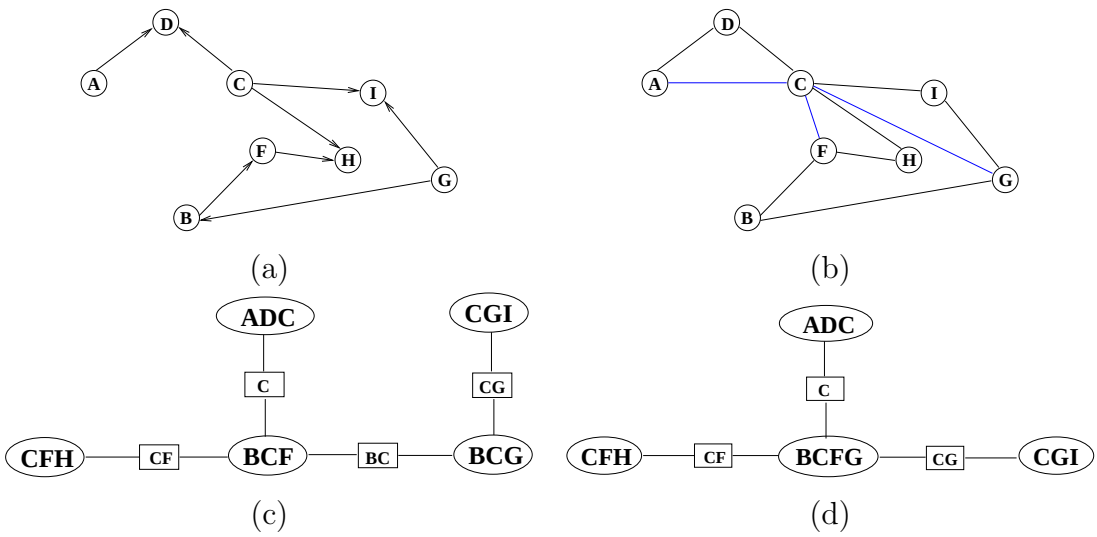


Figure 3.10: Modifications when $B \rightarrow D$ is deleted: (a) BN (b) Moral graph (c) Possible JT and (d) The corresponding MPS tree ($B - C$ is not complete in the moral graph, that comes as a fill-in from an elimination order as $\sigma = \{A, D, H, I, F, C, B, G\}$).

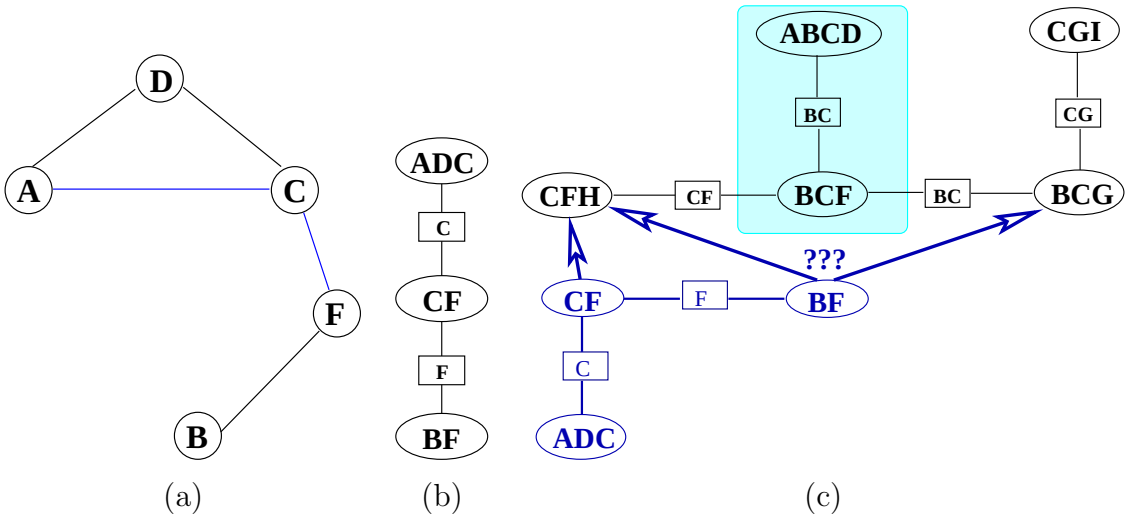


Figure 3.11: IC processing without recursion. Notice that again (the partial) JT and MPS trees coincide here.

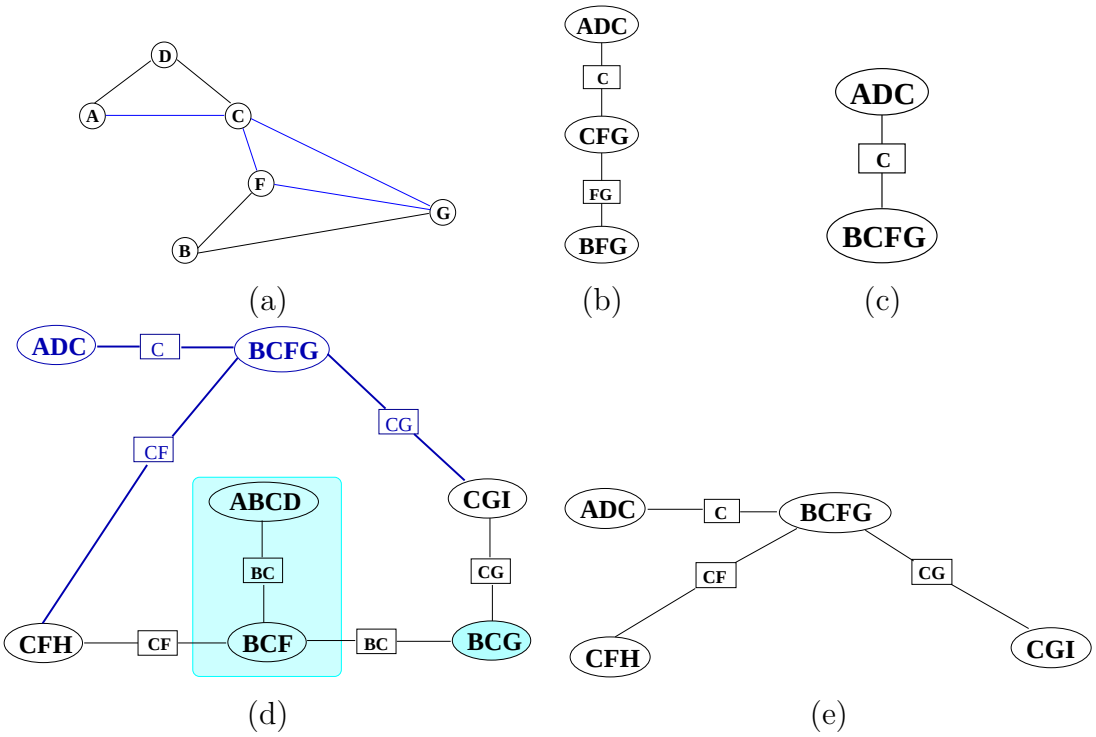


Figure 3.12: IC processing with recursion.

3.2.6 Removing a node

Algorithm 19 marks all MPSs containing X. The algorithm also deletes X from all MPSs and separators containing it in order to obtain the correct set V in step (c) of the second loop of algorithm 15.

As an example, let us consider the removal of variable D from the Asia network. This operation results in the following list of modifications: (remove $E \rightarrow D$, remove $B \rightarrow D$, remove D), which is passed to alg. 15 from the edit mode. Subsequently, the moral link ($E \rightarrow D$) is removed during the first loop of algorithm 15. In the second loop of algorithm 15 the MPSs (DEB) and ($LBES$) are marked. In this case, the effect of algorithm 19 is just to remove D from (DEB). Therefore, we have to re-triangulate $G'^M(\{E, B, L, S\})$, see figure 3.13(c). Part (d) of the same figure shows the obtained tree from this graph. In part (e) the connecting process of the old and new structure is shown, where marked clusters are highlighted by filling them. Finally fig. 3.13(f) shows the obtained result after absorbing non maximal cluster (LE) into cluster (TLE).

Algorithm 19 Algorithm that performs the marking of subgraphs when deleting a node in the Incremental Compilation process.

```
1: procedure RemoveNode(Node X, MPS M_X, MPS M_Y)
2:     Delete X from M_X
3:     Mark M_X
4:     for all Neighbour M_Z ≠ M_Y of M_X do
5:         S ← separator between M_X and M_Z
6:         if X ∈ S then
7:             Delete X from S
8:             RemoveNode(X, M_Z, M_X)
9:         end if
10:    end for
11: end procedure
```

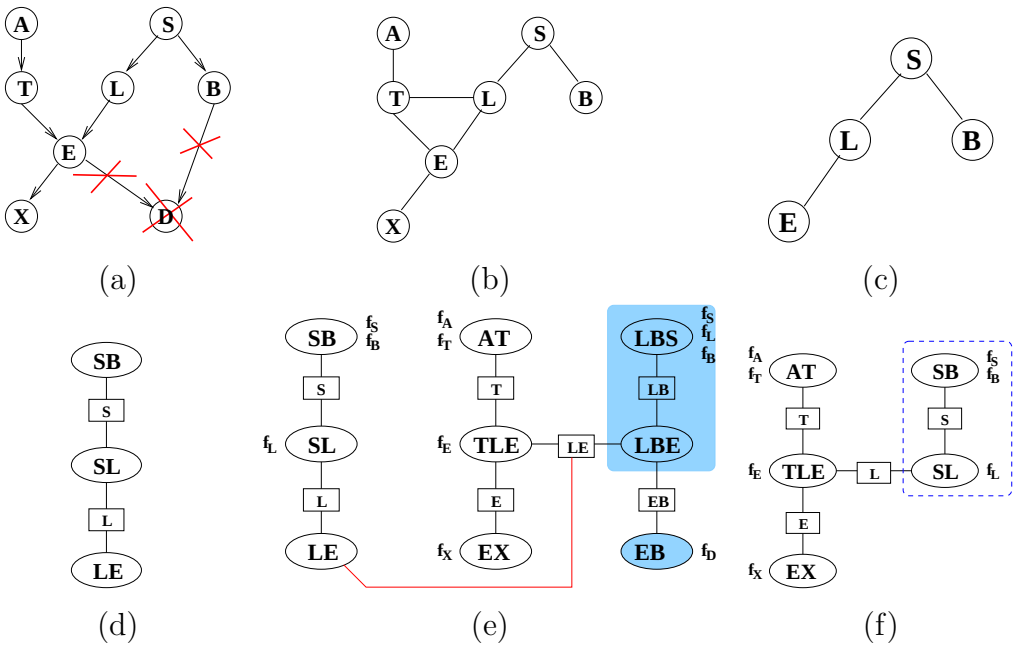


Figure 3.13: MPSD-based incremental compilation of Asia when removing variable D.

3.2.7 Adding a node

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

containing only variable X. In order to maintain single structures (trees) rather than sets (forests), we connect new clusters to the respective trees by picking up any existing cluster and connect the new cluster by an empty separator. In the case that this structure was used to propagate this separator will have capacity 1 in order to store the constant number to be passed from one cluster to the other. The process is detailed in algorithm 20.

Algorithm 20 Algorithm that performs the marking of subgraphs when adding a node in the Incremental Compilation process.

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

As an example, the result of adding a new variable Z to the Asia network is shown in figure 3.14.

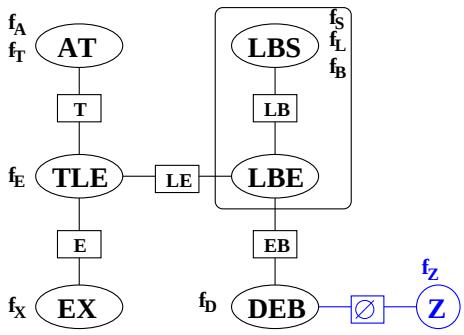


Figure 3.14: MPSD-based incremental compilation of Asia when adding a new variable Z.

3.2.8 Adding a link

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

This is the general idea of the method for adding a link, however, there is a tricky point that should be discussed. Due to the presence of empty separators, it is possible to modify the tree structure after having located M_X, in order to achieve a better (more efficient) structure for our goal. For example if $A \rightarrow Z$ is the link to be added to the structure in figure 3.14, then we will mark all the MPSs in the tree except (EX). However, as M_Z is connected to the tree by an empty separator, we can connect it to MPS (AT) instead. By using this new tree, only MPSs $\{(AT), (Z)\}$ have to be re-triangulated, which leads to a (far) more efficient process.

Algorithm 21 marks the MPSs affected by the addition of $X \rightarrow Y$.

Algorithm 21 Algorithm that performs the marking of subgraphs when adding a (set of) links in the Incremental Compilation process.

```
1: procedure AddLink(LinkList L)
2:     for all Link X → Y ∈ L do
3:         M_X ← the nearest neighbour to M_Y containing X.
4:         if ∃ an empty Separator S (S == ∅) on the path between M_X and M_Y then
5:             Disconnect T_MPD and delete S
6:             Connect M_X to M_Y by an (artificial) Separator containing X
7:         end if
8:         Mark M_X, M_Y
9:         for all M_Z on the path between M_X, M_Y do Mark M_Z
10:        end for
11:    end for
12: end procedure
```

As an example, let us consider the addition of two new links: $A \rightarrow Z$ and $Z \rightarrow X$ to the structure depicted in figure 3.14:

Adding $A \rightarrow Z$: as there is an empty separator in the path between (Z) and (AT), the tree is modified to the one depicted in figure 3.15(a). Algorithm 21 marks MPSs (Z) and (AT). Also, the separator is set to A.

Adding $Z \rightarrow X$: Now, there is no empty separator along the path between (Z) and (EX), so no modification is performed over the tree. The algorithm marks $(Z), (AT), (TLE)$ and (EX) as the MPSs to be re-triangulated.

Remark that the moral link $Z - E$ has been added to G'^M (by algorithm 16).

We get $\{(Z), (AT), (TLE), (EX)\}$ as the set of MPSs to be re-triangulated. The subgraph of G'^M induced by the set of variables in these MPSs $(\{Z, A, T, L, E, X\})$

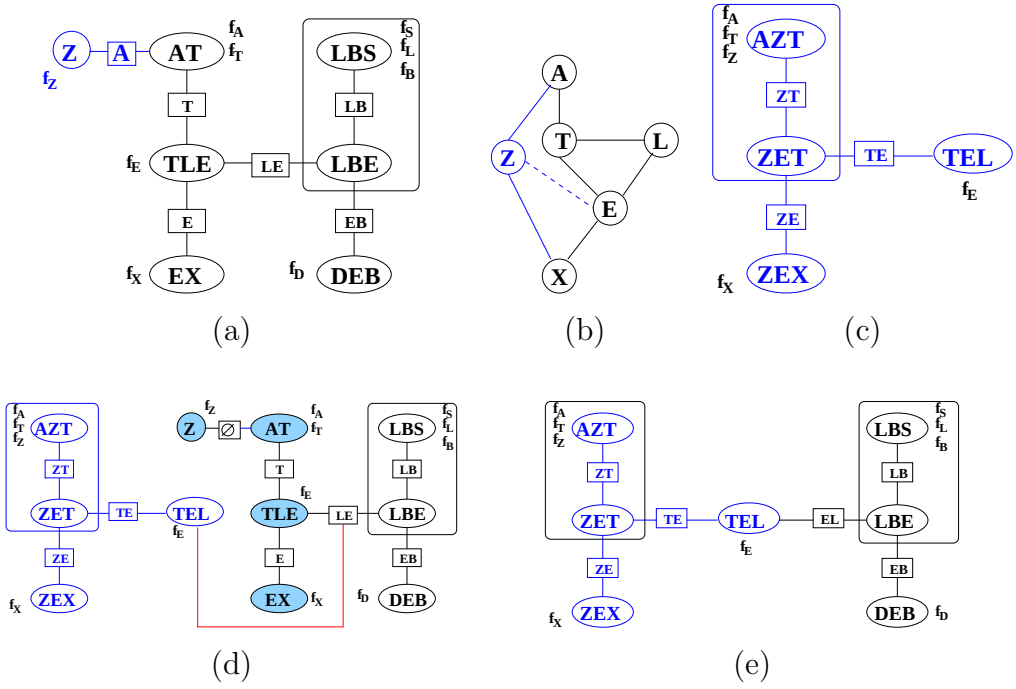


Figure 3.15: Incremental compilation of the structure in figure 3.14 when adding links $\{A \rightarrow Z, Z \rightarrow X\}$.

yields the graph shown in part (b) of figure 3.15, and re-triangulating this we get the tree depicted in part (c) of figure 3.15. Finally, the connecting process is illustrated in fig. 3.15(d) and the final result (after removing marked clusters) is shown in part (e) of the same figure.

3.3 Investigation of this method through a set of distinct networks

The implementation of the method described before has been integrated into the programming code of the Elvira project[3], where several universities participate. The Elvira system [39] is a Java tool (GUI + API) to construct probabilistic graphical models and also to evaluate new algorithms (inference, learning, ...).

In this section we design a series of experiments in order to study the impact of incremental compilation when modifying a Bayesian network. Although the skeleton of experiments presented here is quite similar to the explained in [44], the results

[3] `http://leo.ugr.es/~elvira`

are different. During the last year, we have introduced improvements in basic classes of Elvira system (e.g. a faster search to access a certain variable of the graph) in order to get a better overall efficiency. This has a bearing on the total time for normal compilation as well as in IC, but this gain has been more clear in the first one, since it is applied to longer lists of elements, and reducing these access times is even more crucial when the list of elements is larger. Although the results, as we will next examine, show that IC is advantageous against traditional compilation (and this difference will be more obvious when the changes are reasonably realistic), we feel that IC programming code can also be optimised in order to get even a better performance for it. We expect to work on that enhancement in a near future.

3.3.1 Networks and designed experiments

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

- Ten real complex networks (most of them) taken from the repository[4] of the *Machine Intelligence Group* in Aalborg University and one more, prostanet, taken from [69, 67].

- A set of artificially generated networks. These networks have a sliced-like structure (see figure 3.16) as some times happens in temporal/dynamic and parametric Bayesian networks. What makes these networks interesting is the fact that every two slices (i and $i+1$) are completely separated by the MPS $\{X_{4.i}, X_{6.i}, X_{3.(i+1)}, X_{5.(i+1)}\}$. As $\{X_{1.i}, \ldots, X_{(n-3).i}\}$ and $\{X_{(n-1).i}, X_{(n-2).i}, X_{n.i}\}$ are the two MPSs in each slice, then the number of MPSs in a network is $2 \cdot s + (s-1)$, s being the number of slices in the network. Finally, although the structure of each slice is the same, they can have different optimal triangulations because the number of states for each variable has been randomly generated (by using a Poisson distribution of mean 4 and minimum equals to 2). We have generated ten random networks, termed as RbNxS, with N the number of variables in each slice and S the number of slices.

To avoid the reader to be overwhelmed from too many data and graphics, we have later made a selection of the most representative cases, since there is a common trend. We only reproduce here the experiments carried out over 4 real networks and 4 artificial ones. We have selected this subset in order to have networks with different complexity

[4] www.cs.aau.dk/research/MI/Misc/networks.html

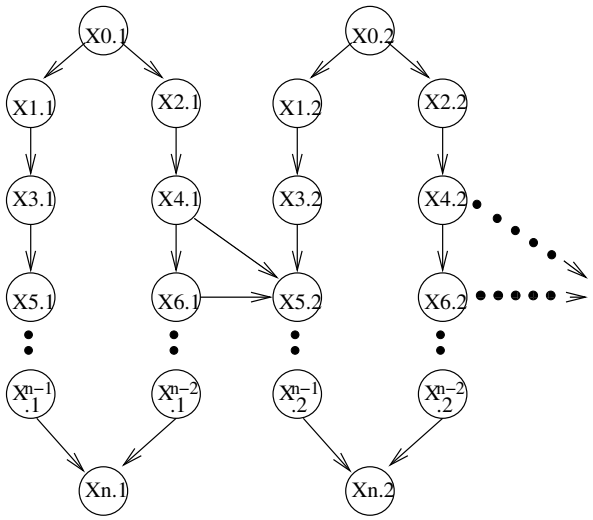


Figure 3.16: Basic structure for the artificially generated networks.

(measured in terms of number of nodes and links). Table 3.1 shows some information about the networks: its name [*Net*], the number of variables [$\#V$], the mean number of states per variable [$\mu(\#St)$] and the number of links/arcs in the network [$\#E$]. Besides, table 3.1 also shows information of interest for the incremental compilation process: the number of links/edges in the moral graph [$\#E_M$], the number of Maximal Prime Subgraphs [$\#S$], the mean number of variables per subgraph [$\mu(\#V_s)$] (plus the standard deviation [$\sigma(\#vs)$]), and the number of variables in the largest subgraph [S_v^*].

Table 3.1: Description of the networks.

Net	$\#V$	$\mu(\#St)$	$\#E$	$\#E_M$	$\#S$	$\mu(\#V_s)$	$\sigma(\#vs)$	S_v^*
Prostanet	47	2.21	81	116	28	3.71	2.88	17
Munin1	189	5.26	282	366	70	4.14	12.61	108
Pigs	441	3.0	592	806	227	3.68	10.09	155
Munin4	1041	5.42	1397	1843	498	3.51	15.40	342
Rb10x5	50	4.36	58	71	14	5.43	2.79	9
Rb10x10	100	3.82	118	146	29	5.38	2.70	9
Rb20x20	400	4.05	438	496	59	8.74	7.42	19
Rb20x50	1000	4.06	1098	1246	149	8.70	7.36	19

3.3.2 General idea for the experimental suite

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

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

3.3.3 Experiment 1: *Random* modifications

The immediate approach for generating modifications would be to randomly select nodes and arcs to add to (or remove from) a network. However, this would probably give rise to unrealistic modifications, as linking nodes which are too far in the graph. Because of this, to generate a list of *realistic* modifications, we propose the following procedure:

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

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

Note that deleting/adding a node involves deleting/adding first/after the incident links (e.g. in figure 3.5.(a) deleting the node L will provoke a previous deletion of both $S \rightarrow L$ and $L \rightarrow E$). So, even selecting a few nodes, the impact of the modification over the network can be significant.

Finally, experiment 1 consists in:

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

In this experiment, the number of variables (numCases) deleted/added is controlled by the parameter n. Depending on the complexity of the network, n has a different meaning: If #V ≤ 100 then numCases=n, otherwise numCases is the n% of #V. Notice that even when $n = 1$ that could imply multiple modifications that will depend on how many children and parents the corresponding nodes present. In order to compare incremental compilation (IC) with traditional re-compilation, we collect different data: time (seconds) and number of nodes/links affected by the modifications.

Tables 3.2 and 3.3 shows the results obtained for n=1 and n=2. The data shown in the table are: the ratio between the time required by re-compilation [t_N] and IC [t_I]; The time required by re-compilation [t_N]; the number of variables [V] and edges [E] modified in the moral graph; the ratio[5] (re-compilation/IC) of variables [V_N^r/V_I^r] and edges [E_N^r/E_I^r] involved in the triangulation process. All the data are on average ($\mu(\cdot)$) over the number of runs carried out. For the number of variables involved in the triangulation process also the standard deviation is shown [$\sigma(\#V_I^r)$]. In our experiments 20 series have been carried out, which gives rise to 40 runs, because every series produces two experiments (modListD and modListA).

3.3.4 Experiment 2: Modifications closer to *customary* usage

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

[5]Note that this ratio is in fact the fraction of the whole network which has to be triangulated in Incremental Compilation.

Table 3.2: Experiment 1 (Random), n=1.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V_I^r}{V_N^r})$	$\mu(\frac{E_I^r}{E_N^r})$	$\sigma(\#V_I^r)$
Prostanet	7.596	0.047	1	6.15	0.32	0.34	9.86
Munin1	2.952	0.564	2	9.25	0.55	0.63	23.16
Pigs	8.769	3.143	4	16.9	0.33	0.34	47.85
Munin4	1.758	19.064	10	37.55	0.36	0.43	30.55
Rb10x5	4.839	0.040	1	3.05	0.23	0.20	3.28
Rb10x10	8.355	0.160	2	6.1	0.22	0.19	6.02
Rb20x20	12.103	2.228	4	10.25	0.20	0.18	12.59
Rb20x50	17.663	17.624	10	25.9	0.19	0.17	32.59

Table 3.3: Experiment 1 (Random), n=2.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V_I^r}{V_N^r})$	$\mu(\frac{E_I^r}{E_N^r})$	$\sigma(\#V_I^r)$
Prostanet	6.934	0.047	2	10.85	0.42	0.43	9.95
Munin1	1.092	0.855	4	17.05	0.60	0.67	3.61
Pigs	1.411	3.236	9	35.4	0.40	0.40	10.41
Munin4	1.492	19.339	21	77.6	0.42	0.47	33.16
Rb10x5	2.980	0.038	2	5.9	0.40	0.35	5.88
Rb10x10	4.317	0.153	4	12.25	0.41	0.35	9.82
Rb20x20	6.542	2.169	8	21.15	0.36	0.33	21.31
Rb20x50	9.227	17.284	20	52.15	0.35	0.32	57.42

in that region.

To reflect this more realistic behaviour we change the manner in which the nodes to be modified are selected. First, we randomly select a *leaf* node X_1 from the network and all the nodes linked to it are included in a set N. This set, that will incrementally receive new elements, is the container of the next nodes candidates to be selected. Then, at stage i the next node X_i is randomly selected from N, and all the neighbours of X_i are added to N. In this way all the modifications are concentrated in the same region of the network. We have denoted this experiment as *neighbour* while experiment 1 is termed *random*. Tables 3.4 and 3.5 show the results of this experiment.

Table 3.4: Experiment 2 (Neighbour), $n = 1$.

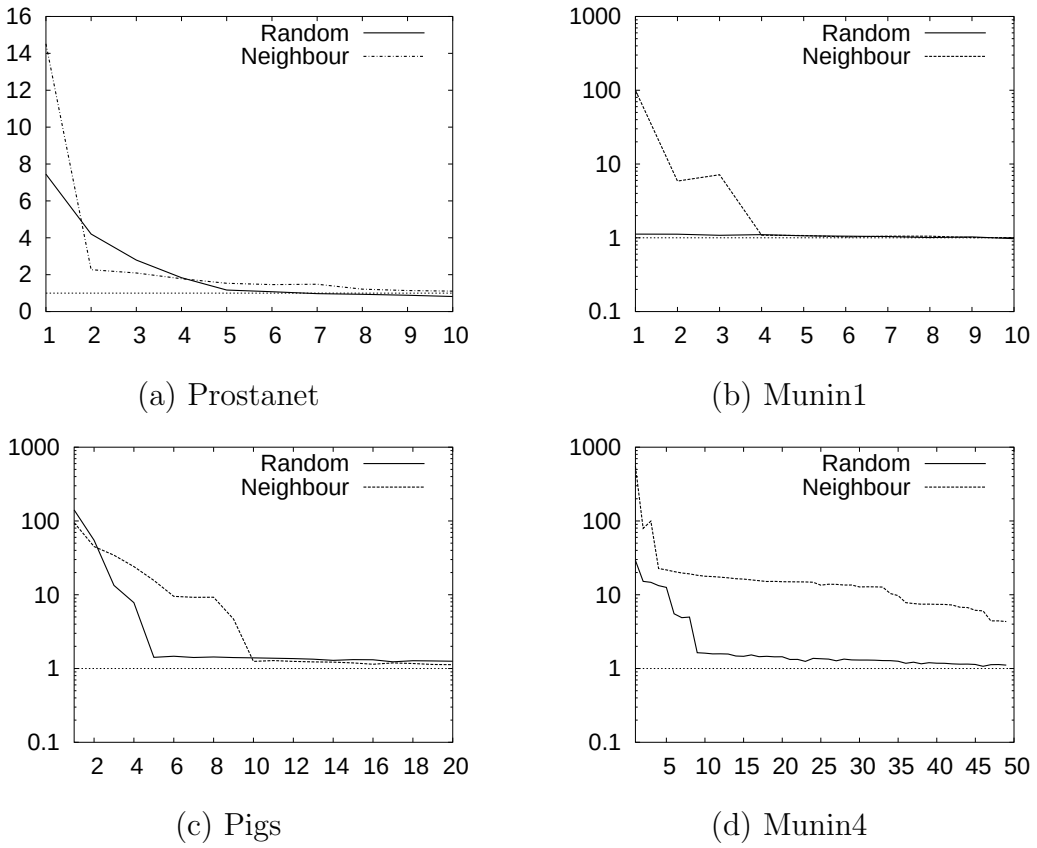
Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V_I^r}{V_N^r})$	$\mu(\frac{E_I^r}{E_N^r})$	$\sigma(\#V_I^r)$
Prostanet	14.136	0.048	1	1.75	0.15	0.14	7.31
Munin1	3.249	0.850	2	9.15	0.57	0.65	17.35
Pigs	20.909	3.111	4	31.85	0.33	0.33	60.43
Munin4	16.001	19.075	10	31.75	0.22	0.26	149.43
Rb10x5	5.463	0.039	1	3	0.22	0.18	1.99
Rb10x10	16.651	0.165	2	4	0.11	0.09	2.81
Rb20x20	44.711	2.285	4	6	0.05	0.05	3.97
Rb20x50	164.202	18.428	10	12.2	0.02	0.02	7.93

Table 3.5: Experiment 2 (Neighbour) ,$n = 2$.

Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	V	E	$\mu(\frac{V_I^r}{V_N^r})$	$\mu(\frac{E_I^r}{E_N^r})$	$\sigma(\#V_I^r)$
Prostanet	2.222	0.046	2	15.65	0.50	0.47	7.81
Munin1	1.079	0.856	4	16.35	0.59	0.67	4.08
Pigs	2.842	3.174	9	61.7	0.40	0.39	32.56
Munin4	13.587	19.750	21	67.65	0.24	0.27	158.85
Rb10x5	5.626	0.039	2	4	0.21	0.18	2.48
Rb10x10	12.941	0.159	4	9.05	0.14	0.12	6.00
Rb20x20	46.252	2.300	8	10	0.04	0.04	5.99
Rb20x50	76.699	18.705	20	28.8	0.04	0.03	16.10

3.3.5 Experiment 3: Impact of the number/size of modifications on IC performance

In the previous experiments we have modified the network by using $n = 1$ and $n = 2$ which in some cases produce a big impact on the resulting model. Also, many statistics about the process have been collected. In this experiment we ran the process described in experiments 1 and 2, but we only focused on the ratio $\frac{t_N}{t_I}$. On the other hand, we ran the experiment for numCases = 1, 2, ... (that is, the parameter n is not used in this experiment[6]). Figures 3.17 (real networks) and 3.18 (artificial networks) show the results of this experiment averaging over 10 different runs, where numCases is represented in axis X and the ratio $\frac{t_N}{t_I}$ is represented in axis Y. Note that in some cases logarithmic scale has been used in axis Y so as to offer a finer visualisation.

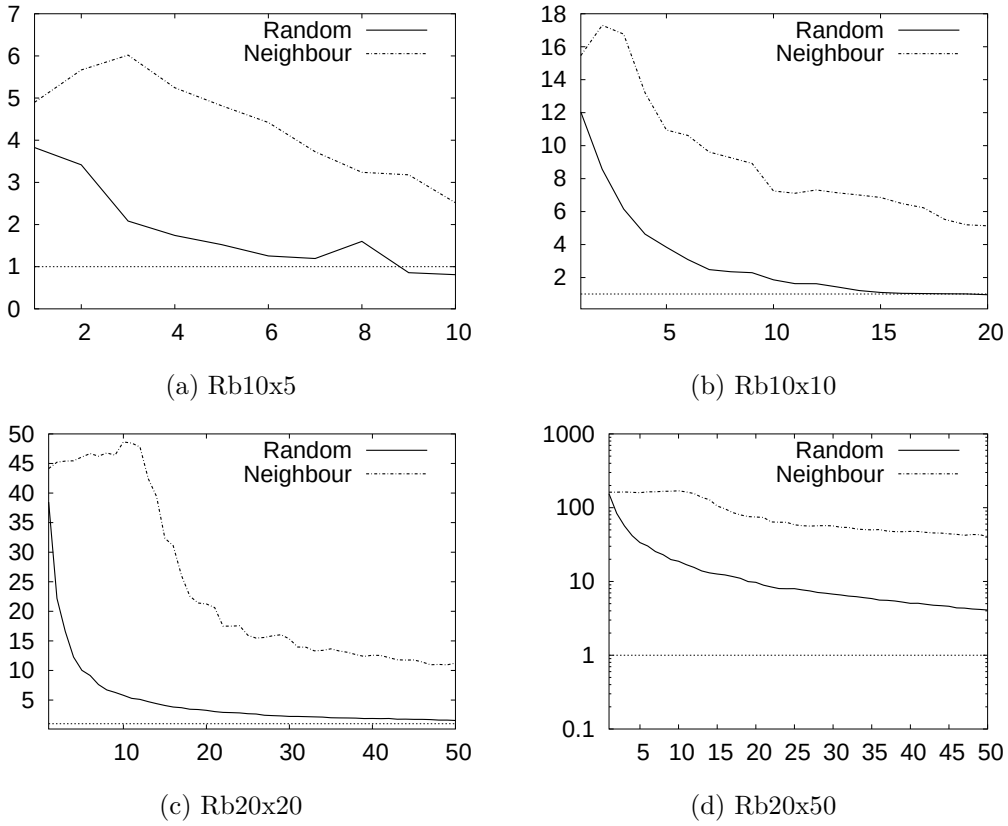


(a) Prostanet (b) Munin1

(c) Pigs (d) Munin4

Figure 3.17: Impact of numCases over the ratio t_N/t_I for the *real* networks.

[6]Notice that for example in network Munin4 $n = 1$ implies numCases=10.



(a) Rb10x5 (b) Rb10x10

(c) Rb20x20 (d) Rb20x50

Figure 3.18: Impact of numCases over the ratio t_N/t_I for the *rhombus* networks.

3.3.6 Experiment 4: Addition vs Deletion changes for IC

In this experiment we want to compare the effect of the two types of modifications (deletion and addition) when performing incremental compilation. To do this, we base our study on experiments 1 and 2, but now we show the ratio t_N/t_I separately for runs involving deletions (type D – modListD) and runs involving additions (type A – modListA). Table 3.6 shows the results obtained for a representative subset of the networks used in this paper, where we distinguish between *random* and *neighbour* modes as usual. We should note that, due to the experiment design, in the deleting (D) *phase* the network is decreasing incrementally in size, because we are deleting elements (nodes together with their incident links) while in the second stage (adding - A) the network will grow again until adopting the original structure.

Table 3.6: Results for modifications of type: deletion (D) and addition (A).

	Random				Neighbour			
	D		A		D		A	
Network (n)	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$
Munin1 (1)	7.183	0.820	6.611	0.668	5.383	0.823	1.114	0.877
(2)	1.088	0.811	1.095	0.897	1.086	0.814	1.073	0.898
Pigs (1)	16.055	3.063	1.483	3.223	30.481	2.997	11.336	3.225
(2)	1.368	3.093	1.453	3.379	4.271	2.942	1.413	3.407
Rb20x20 (1)	14.813	2.134	9.394	2.316	56.980	2.249	32.442	2.321
(2)	8.112	2.010	4.972	2.328	62.83	2.262	29.674	2.338
Munin4 (1)	1.811	19.028	1.704	19.100	22.034	19.086	9.968	19.063
(2)	1.550	18.834	1.434	19.843	19.568	19.489	7.606	20.011

3.3.7 Experiment 5: Influence of IC when creating potentials

Compilation of a BN is the process that takes a network as input and produces an initialised join tree, that is, a join tree over which inference can be carried out. Depending on the inference method, the meaning of an initialised join tree differs. Thus, if Lazy Propagation is used [81] it is enough to build the join tree and to establish the assignment of the network probability families (conditional probabilities) to the cliques in the join tree. This is just the process we have measured in experiments 1 to 4. However, if lazy propagation is not used (which is the case for most Bayesian network tools), then the potentials associated to each clique have to be initialised as the product of the probability families assigned to it. In this experiment we analyse the effect of using IC when potentials are initialised as probability tables (the usual representation, e.g. [74]). Table 3.7 shows the data for this experiment with $n = 1$ and $n = 2$. In this case the measures times T_N and T_I include compilation (triangulation + JT construction) together with potentials initialisation.

We have only included one of the artificial networks: Rb20x20. This is because they are not our particular aim in this case as their structure provokes that nearly all the cliques contain only one family. So, there is barely multiplication of tables and we would be in a situation very close to that in experiments 1 and 2. For real networks, we skipped Munin1 due to the memory resource required by the large state space of its probability tables.

Table 3.7: Results for experiments initialising tables ($n = 1$ and $n = 2$).

	Random		Neighbour	
Network	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$	$\mu(\frac{t_N}{t_I})$	$\mu(t_N)$
Prostanet (1)	11.859	0.084	18.471	0.086
(2)	5.986	0.084	3.480	0.082
Pigs (1)	11.740	7.272	56.144	7.158
(2)	2.255	6.659	2.204	6.520
Rb20x20 (1)	11.113	2.543	41.016	2.585
(2)	5.996	2.455	41.795	2.572
Munin4 (1)	1.249	156.525	26.436	177.676
(2)	1.191	146.357	30.353	169.48

3.4 Analysis from the experiments

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

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

- When modifications are selected randomly (exp. 1) the gain provided by IC increases with the complexity of the BN and the MPD (number of subgraphs and number of variables per subgraph) and decreases with the modified portion of the network (parameter n).

- When a real behaviour is simulated (*neighbour*) the previous observation also holds and now the gain obtained by IC is severely increased. For example, with Munin4 and $n = 2$, even though a big fraction ($\simeq 25\%$) of the BN is affected by IC, our method is 13 times faster than Non-IC.

- As it has been pointed out the larger the network is the bigger the gain is. Moreover, it is important to notice that precisely these large networks are those that require more CPU time to be compiled. As an example, a speedup of 1.09 in Munin1 means that 0.78s are needed instead of 0.86s while in Munin4 the

speedup of 1.76 means that less than 11s are needed instead of 19s. For that reason, larger networks are the ideal target for IC.

- In experiments 1 and 2 a moderate number of modifications has been considered. The purpose of experiment 3 is to illustrate how IC behaves as a function of the number of modifications. From the graphics shown in figures 3.17 and 3.18 we can observe a huge speedup obtained when only a few variables (and their links) are modified. Remark that for large networks (Munin4) even modifications based on up to 30 variables yields a considerable speedup.

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

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

- In exp. 5 we go beyond structure construction and we also initialise the clique potentials. This procedure involves the multiplication of probability tables which is a time consuming process for large tables such as in Munin4. That is why recompiling with $n = 2$ Munin4 needs 6s (IC) vs 3 minutes (Non-IC). IC also improves its speedup with respect to Non-IC in Pigs network, however the improvement is smaller than in Munin4. This is caused by the fact that in Pigs all the probability tables are rather small (3 variables $\times$ 3 states, at most).

- The results obtained for small networks[7] of around 40-50 nodes are quite particular when the BN only presents one or just a few leaf node(s). In the *neighbour* case the selected node is almost always the same and the next node to delete will be its parent, which normally belongs to a big clique. So it is curious that under these circumstances *random* experiments avoid this situation and could give better results.

[7]Not all reproduced here, but tested in our whole experimentation.

3.5 Main conclusions and further work

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

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

The main algorithm IncrementalCompilation is constructed such that it can be activated at any time. This enables the procedure to operate both in simple mode, where it is called for each simple modification to the BN, and in batch mode, where several modifications are processed simultaneously. Thus, a situation similar to inference, where inference can be performed either for each piece of evidence or for a group of findings, can be obtained. This feature is seen in most tools for construction and execution of BN models.

After a first methodological work to devise this technique [43], we started investigating the programming of the given algorithms to test their practical impact. Then, in the second part of this chapter an experimental evaluation of Incremental Compilation of Bayesian networks has been presented, completing the first algorithmic part [44]. To tackle this practical work, we designed first a set of experiments that could be representative for distinct real cases and taking into account all kind of modifications and particular conditions that could make IC easier or harder to perform. From the analysis of the experiments carried out we can conclude that IC is an advantageous method with respect to compilation from scratch, in particular when the network is large and slightly modified. But these two particular conditions were the ones we were really interested to solve, since large networks was the problematic case for a recom-

pilation and the changes done on them (when modelling or other iterative tasks) are quite usually restricted to a little amount of the network located in a limited area.

We reckon that this method has been proved of interest even in networks having a (by far) non-uniform MPS Decomposition. Since, this is finally a *divide and conquer* approach it is expected its efficiency grows if the division is more balanced. It is for that reason that the speedup increases enormously when the network has a *nice* MPSD.

Apart from an optimisation of the programming code in general, we would like to characterise theoretically certain *unusual* situations that need extra processing tasks. By this, we could save the computational time needed to guarantee that the method is always correct. We believe that it is possible to detect when to apply these *secure* processes and that characterisation will allow us to escape from the unnecessary use of resources that occurs in some occasions.

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

Also as a future work it would be possible to investigate about the extension of this method from discrete Bayesian networks to other kinds of graphical models, such as CG-models and influence diagrams.

We would like to point out that the Incremental Compilation task can be interpreted as well as a modular collaborative distribution of the network that is obtained directly from it. That is, we partition the network in subgroups that correspond to the MPSs. In the literature we find many attempts to do this modular division from the opposite point of view, that is, constructing a total and larger network from smaller items that might be seen as subgroups or subnetworks. This is another incremental perspective, were the elements are accumulated, connected and integrated to form bigger structures. These conceptions are usually located in the framework of *Knowledge Engineering* and the *Object Oriented* philosophy. From this basis, we will devote next chapter to comment the main proposals for diverse modular structures, and to analyse the connections between IC and them which can be of great utility for certain purposes, as the inference process.

Chapter 4

Revision on *modular* Bayesian structures

Divide each difficulty into as many parts
as is feasible and necessary to resolve it.
René Descartes. (1595–1650)
French mathematician, scientist and philosopher.

4.1 Introduction

Recently, there has been a great interest in obtaining *modular* Bayesian Networks. There have been different approaches to do so. The main motivation of all of them coincides. When tackling complex problems a single BN presents many difficulties:

- The modelling task is quite hard, since we need to consider all the factors involved in the problem at a unique stage.
- Once we have obtained this total model, the resultant inference construction seems to be unmanageable or it just takes too much time and space for computation.
- Afterwards, for future modifications, additions or removals, it is again necessary to face up to the whole representation, even when many parts should not be affected by these changes. Thus, it is muddling to work with big structures, as the overall view sometimes do not let us be able to detect the elements involved in one concrete part of the problem.

There have been several trials in order to deal with these complex, but very usual problems. Since normally Bayesian networks turned to be inadequate as a general representation language for large and complex domains [82], a combination of *network fragments* in order to form problem-specific models to reason about particular applications was discussed in [72]. This work presented a framework to support automated model construction from a knowledge base expressing generic probabilistic relationships. These foundations have also given rise to many further research fields and applications, such as network construction [83], learning [85] or the possibility of working with partially specified Conditional Probability Tables [84].

Another contemporary, yet independent, approach are the Object-Oriented Bayesian networks (OOBNs) by Koller and Pfeffer [66]. Both frameworks work in the same direction, but using different tools. They share some features such as the incremental construction from simpler structures (let us call them fragments or objects) to complex ones. However, they differ not only in the usage of their representation, but also in their nature: network fragments build complex models procedurally while OOBNs use a declarative object-oriented representation language. As a result, this latter approach allows the organisational structure of a model, in particular the encapsulation of objects and the reuse of OONFs within a model, to be expressed explicitly and utilised by the inference algorithm. This object-oriented framework defines a basic unit, which is the *object*, as a collection of properties (*attributes*) associated to some entity in the problem domain. These properties will belong to a certain and previously defined type. And there is an important distinction between what they call input (like parameters passed to an object), output (as the *information* produced by the object) and encapsulated attributes (internal to the object and unknown to the *outside*). An example of this framework is given in figure 4.1. Also, in figure 4.2 we can see an overview of the interconnection between elements. A more detailed study of this latter framework can be found in [102]. It presents some common features with another OOBN approach that will be seen in detail later in this chapter.

Apart from the previous brief review of various relevant proposals in the field, we will focus on two other concrete frameworks: Multiply Sectioned Bayesian Networks (MSBNs) by Xiang [133] and the Object Oriented Bayesian Networks (OOBN) approach by Bansgø and Wuillemin [4]. The first one is particularly interesting for us, since we find close similarities with Incremental Compilation procedure in a way that both can be benefitted from each other and collaborate. That will be more deeply pre-

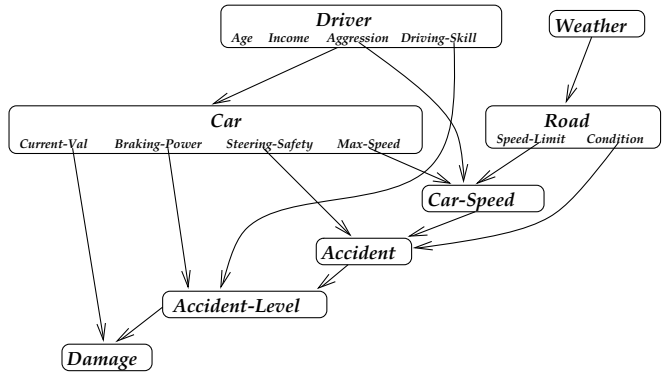


Figure 4.1: OOBN model for a car accident.

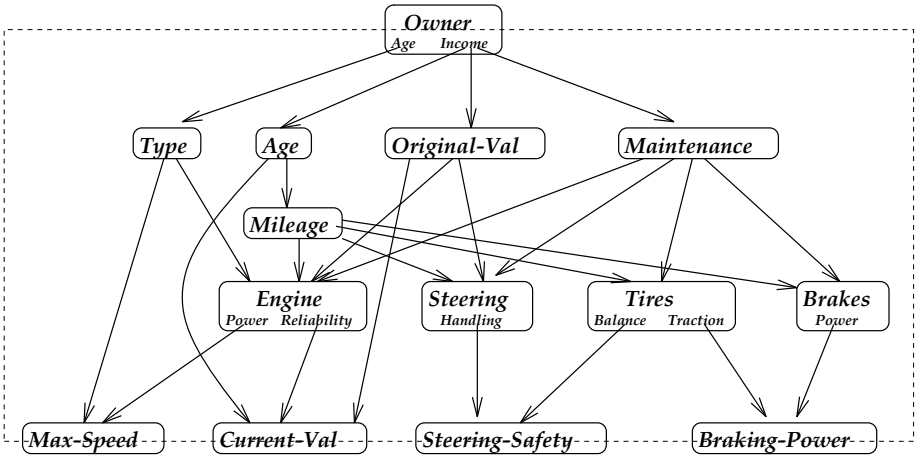


Figure 4.2: Interconnection model in an example OOBN modelling for a car.

sented in the following sections. On the other hand, OOBNs[1] inference has also taken advantage of the Incremental Compilation technique collaborating in the achievement of the so-called Plug & Play OOBNs [6] as section 5.4 will describe. In addition OOBNs and MSBNs are also closely related, and there exist techniques for mutual translation, which can also be quite useful for our proposals.

But again, even if they present analogies their view and nature are clearly separated. MSBNs can be understood as an extension of the *traditional* Bayesian network as a single-agent paradigm to the multi-agent scope. On the other hand OOBN, as well as the first commented frameworks, can be located closer to the discipline of Software

[1]From now we will refer to OOBNs considering the framework presented by Bansgø and Wuillemin.

Engineering. Besides, in MSBNs the partitioning of the undertaken problems is natural, and the hidden or internal information is originally like that. On the contrary, OOBNs are normally suitable for exploiting repetition, for reusability or for dynamic structures.

4.2 Multiply Sectioned Bayesian networks: basics on MSBN

Multiply Sectioned Bayesian Networks have been mainly developed by Xiang. The first work about this structure is covered in his Ph. D. thesis [129]. In this thesis we can find the theory for MSBNs and junction forests as well as its implementation in WEBWEAVR (see figure 4.3), for which version IV is about to be released. The first version of this software was used in order to develop PAINULIM, a real system for neuro-muscular diagnosis [130]. But an overall of the research done over this subject and a more complete work is on his recent book [133].

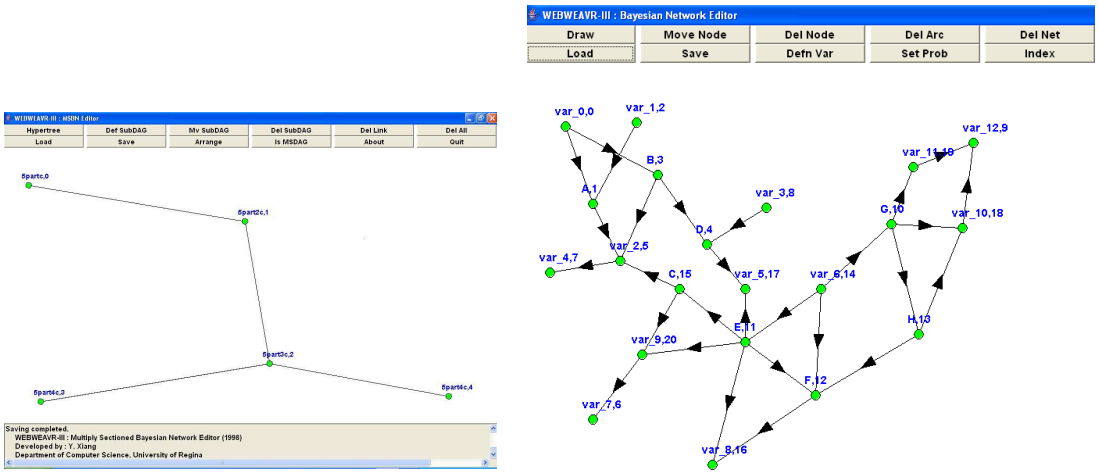


Figure 4.3: Screen shot of the software tool WEBWEAVR-III. On the left: the MSBN editor window showing the example MSB network *5part*. On the right side it is shown the BN corresponding to one of the agents of this MSBN.

The idea behind the MSBNs is the possibility of obtaining the natural modularity of many problems we commented above. Xiang sees Bayesian networks as a graphical model used by an intelligent system to be able to reason. What he proposes is that we can have a multi-agent system that combines different parts of a problem. These parts

can be Bayesian networks that together form an MSBN. Basically, MSBNs present these main features:

1. A set of Bayesian networks, called *subnets*, collectively define a BN.
2. Interface between subnets that render them conditionally independent.
3. Top level structure as a hypertree.
4. This hypertree is compiled into a Linked Junction Forest (LJF) for inference.
5. Coherent inference operations are defined for a LJF.

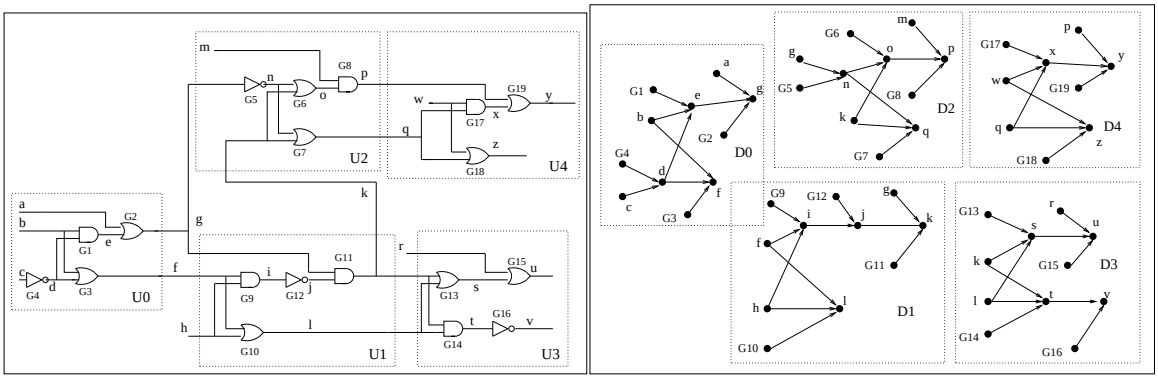
These main points can help for a global overview of the theory for MSBNs, it is nevertheless necessary to go further in their description in order to study the problem we try to undertake. And that is the purpose of the following subsections.

At this point we refer to figure 4.4.(a), that describes a system for diagnosis of faulty gates, the next figure (4.4.(b)) is the resultant MSBN that this example yields. This can give a first intuitive glimpse to the problem we try to deal with. Of course, this is a very simple example, but that illustrates quite well the type of situations we would like to model and it is easy to follow, but we should remark that MSBN are thought to work with more complex domains. Let us see figure 4.4.(a). We find a digital system, with some signals (lower case letters) and some digital gates (OR, AND and NOT gates) we name as Gi. In this particular problem, there is a natural partitioning since the whole circuit is supposed to be composed of different parts that might come from various circuit vendors. So, every subsystem may be not only different in nature, but also unknown for the rest, and this lack of information could even be introduced deliberately in order to hide the specifications from other (*market competitor*) vendors.

The dotted line areas indicate the different subsystems $U0$ to $U5$. We can see there are some shared signals as f between $U0$ and $U1$ or g between $U0$ and $U2$. This will be translated into the communication elements, but there will be some other components only known by the subsystem that contains them. For instance, signal m, or gate $G6$ are internal information for only subsystem $U2$.

The associated set of subnets could be the one depicted in figure 4.4.(b). If we look as just one of them it is a *traditional* BN, where both signals and gates are represented by nodes in the network, and for every gate we have the conjunction of the gates and input signals will affect the output signal, represented by the links. For example, in the subnet $D0$, the subgraph formed by a,e,g and $G2$ comes from the or-gate $G2$ in

the previous figure. We can see that the output signal (g) depends on the input signals a and e, and also on the gate $G2$, that can work properly or otherwise it could be faulty. This diagnosis of faulty-gates is what the system tries to determine. But the particularity of these nets is that they have some nodes in common as the system presented some elements in common.



(a) (b)

Figure 4.4: (a) Digital circuit for detecting faulty gates. (b) Corresponding MSBN.

4.2.1 Basic Assumptions for MSBNs

As we have already seen MSBNs is a generalisation of BNs taken them to a multi-agent paradigm. It is obvious that this translation will require some new concepts. Firstly, we are going to define some of these concepts and we will also sketch out the set of basic assumptions that Xiang outlines in his book. What he makes is a kind of travel where he shows those conditions that seem to be desirable in order to have an efficient and safe way of treating these structures.

An *agent* will be an element of the system that deals with a concrete part of the overall system (subsystems), interacting with the rest of the agents, but containing its own information that will be private and hidden to the rest of the system. Agents will communicate through common components. Two neighbours agents share one or more components that will constitute their *interface*. In this approach each agent can be seen separately as a small Bayesian network, subset of the global one, which is called *subnet*. These subnets must be seen from the point of view of *cooperating agents*.

Let us see now the five basic assumptions we need to take:

Basic Assumption 1 *Each agent's belief is represented by probability.* □

What remains behind this statement is that we must be able to perform belief updating in an **exact** way (at least theoretically it would be possible). Also, if we assume that the total domain V is populated by a set of agents $\{A_i\}$, each agent A_i has knowledge (structural knowledge and also numerical by this probability representation), over the subdomain $V_i \subset V$.

Basic Assumption 2 *An agent A_i can in general influence the belief of each other agent through direct or indirect communication but can communicate directly to another agent A_j only with $P_j(V_i \cap V_j)$ where $V_i \cap V_j \neq \emptyset$.* □

This second assumption considers that the communication between adjacent agents can be done directly and only with a concise message: a belief table over the variables both of them share. As said before an agent A_i covers a subdomain V_i. In order to communicate to other agent it is needed a common subset $V_i \cap V_j = I \neq \emptyset$. The other variables $(V_i \setminus I)$ will be private for the current agent (unless they belong to the interface of A_i to another agent A_k, $k \neq j$).

Basic Assumption 3 *A simpler agent organisation (as a subgraph of the communications graph) is preferred.* □

With this third assumption he recommends that a tree-organisation for agents communication should be adopted. This result is already known from the single-agent field research, as cluster trees were normally preferred to clusters graphs for belief updating. That structure will help for a better cooperation process. It will facilitate the messages flow.

Basic Assumption 4 *A Directed Acyclic Graph (DAG) is used to structure each agent's knowledge.* □

This requirement is necessary on the sake of efficiency. To encode domain dependence correctly graphical structure should be an I-map. We need to guarantee that restriction. Also, in order to organise agents, we also have to decide the conditions for the *bounds* between agents. So a separation criterium will be covered here (d-sepsets).

Basic Assumption 5 *Within each agent's subdomain, a Joint Probability Distribution (JPD) is consistent with the agent's belief. For shared variables, a JPD supplements an agent's knowledge with others'.* □

This final postulate enforces cooperation among agents and interprets the JPD as the *collective belief* of all agents.

Basically, the way in which Xiang presents all these assumptions follows the scheme depicted below:

- Firstly, Basic Assumption (hereafter BA) 1 comes from our interest about using the Probabily Theory, mainly because of the origin of MSBNs, based on the original BNs.
- Then, BA 2 is due to the inheritance of this new theory from the communication between clusters in the traditional methods of inference over BNs.
- At this point we can state that if we have a communication graph H of a multi-agent system presenting both BA 1 and 2, then H is connected.
- BA 3 just guarantees that the communication process between agents will be relatively simple as trees present the desirable features for it (we can think about the junction tree for propagation, for instance).
- In this reasoning line, a multi-agent system that observes the first third BAs can use its tree organisation of agents for communicating beliefs. And we could therefore make a sort of junction tree organisation of agents for this purpose.
- If we wish to work independently with every agent, then we should pursue that the interface in this tree organisation renders subdomains in the two induced subtrees conditionally independent. Remember that we call *interface* the shared variables between two adjacent agents.
- Since the representation using a graphical model seems to be quite suitable when the domain is complex, we assume that a DAG will be used to structure the agent's knowledge about a subdomain. As we already said BA 4 is not a necessary characteristic, but quite convenable because knowledge acquisition can be performed quite compactly using this DAG structure.
- So, in this stage of the analytical thread, if a multi-agent system observes BAs from 1 through 4 we can assure that each subdomain V_i is structured as a DAG over V_i and besides the union of these DAGs is a connected DAG over V. This statement as well as many of the results we are presenting here are written in Xiang's book as propositions with their corresponding proofs.

- From the previous points we are just ready to present the concept of **d-sepset** (recall definition of d-separation in chapter 1, def. 5)

  Definition 23 Let $G_i = (V_i, E_i)(i = 0, 1)$ be two DAGs such that $G = G_0 \sqcup G_1$ is a DAG. A node $x \in I = V_0 \cap V_1$ with its parents $pa(x)$ in G is a **d-sepnode** between G_0 and G_1 if either $pa(x) \subseteq V_0$ or $pa(x) \subseteq V_1$. If every $x \in I$ is d-sepnode, then I is a **d-sepset**. □

  Figure 4.5 shows an example where the condition either $pa(x) \subseteq V_0$ or $pa(x) \subseteq V_1$ is necessary to obtain the d-sepset.

- Thus, the next requirement will be that each agent interface Z should be a d-sepset. Until here, we have described a multi-agent dependence structure at two levels: the agent modelling and the cooperative system organisation. Now the tool that joins both levels is the *hypertree* notion.

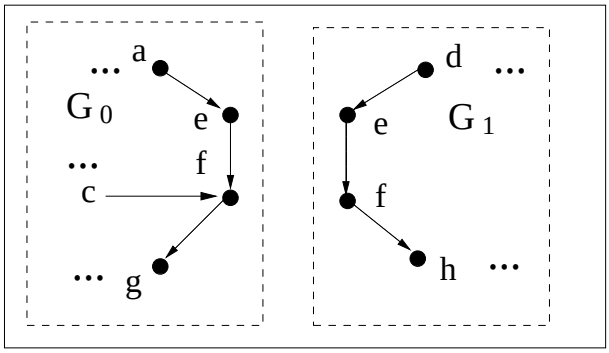


Figure 4.5: An agent interface e,f that is not a d-sepset. G_0 and G_1 represent agents and dots additional variables we just don't need to show.

Definition 24 Let $G = (V, E)$ be a connected graph sectioned into subgraphs $\{G_i = (V_i, E_i)\}$. Let the $G_i s$ be organized as a connected tree Ψ, where each node is labelled by a G_i and each link between G_k and G_m is labelled by the interface $V_k \cap V_m$ such that for each i and j, $V_i \cap V_j$ is contained in each subgraph on the path between G_i and G_j in Ψ. Then Ψ is a **hypertree** over G. Each G_i is a hypernode, and each interface is a hyperlink. □

Let us see the hypertree of figure 4.6.(b). It comes directly for the graph G in 4.6.(a).

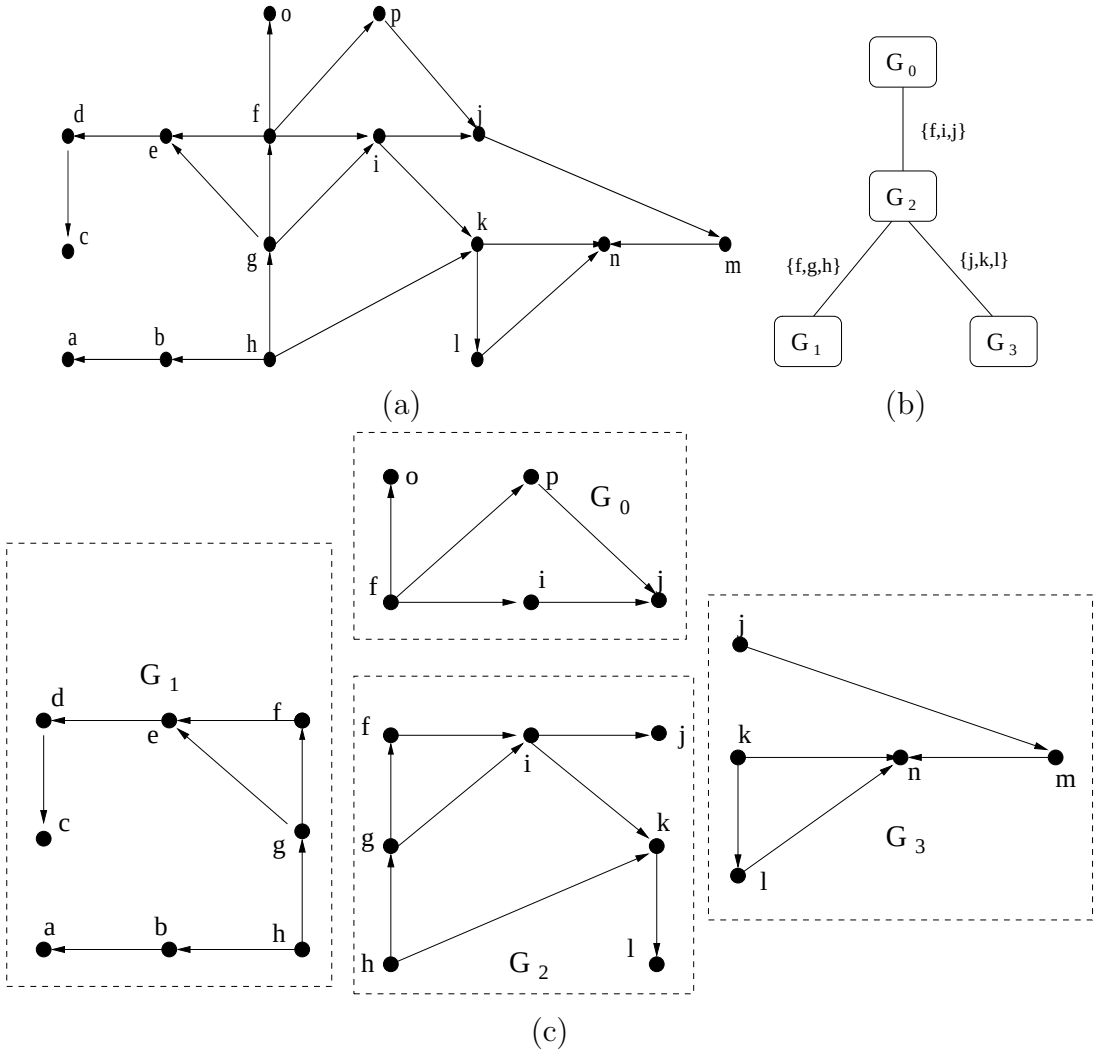


Figure 4.6: (a) A graph G. (b) The hypertree Ψ over the Graph G. (c) The four subgraphs G_0 to G_3 in which G has been sectioned and that yield hypergraph Ψ.

In some cases the d-sepset definition (Def. 23) can fail when talking about nodes shared by more than two subgraphs. For that, Xiang after a deeper analysis of the situation extends the definition of **d-sepset** as follows:

Definition 25 Let G be a directed graph such that a hypertree over G exists. A node x contained in more than one subgraph with its parents $pa(x)$ in G is a **d-sepnode** if there exists one subgraph that contains $pa(x)$. An interface I is a **d-sepset** if every $x \in I$ is a d-sepnode. □

- In a next step, it is proved that if Ψ is a hypertree over a directed graph $G = (V, E)$, for each hyperlink I splits Ψ in two subtrees over $U \subset V$ and $W \subset V$, respectively $U \setminus I$ and $W \setminus I$ are d-separated by I if and only if each hyperlink in Ψ is a d-sepset.

- From all this, the concept of *hypertree multiply sectioned DAG* or MSDAG (see def. 26) comes, having the certainty that a multi-agent system presenting BAs 1 to 4, can be structured as a hypertree MSDAG

  Definition 26 A **hypertree MSDAG** $G = \sqcup_i G_i$, where each $G_i = (V_i, E_i)$ is a DAG, is a connected DAG such that (1) there exists a hypertree Ψ over G, and (2) each hyperlink in Ψ is a d-sepset. □

- We then can say that the MSDAG is the dependence structure equivalent to the DAG structure in a BN. What we finally want to compute is the belief, so we need in some extent to be able to know the JPD from each agent's belief, and we also wish to have consistent information. For that, the fifth and final BA is presented.

  This assumption attempts to integrate independently the construction of agents into a coherent system.

- Going through all these BAs we have finally reached the definition of a Multiply Sectioned Bayesian Network:

  Definition 27 An MSBN M is a triplet (V,G,P) where

  - $V = \cup_i V_i$ is the **total universe** where each V_i is a set of variables called a subdomain.

- $G = \sqcup_i G_i$ (a hypertree MSDAG) is the structure where nodes of each subgraph G_i are labelled by elements of V_i. For each x, exactly one of its occurrences (in a G_i containing $x \cup pa(x)$) is assigned $P(x|pa(x))$, and each occurrence in other subgraphs is assigned a uniform potential.
- $P = \Pi_i P_i$ is the JPD, where each P_i is the product of the potentials associated with nodes in G_i. Each triplet $S_i = (V_i, G_i, P_i)$ is called a subnet of M. Two subnets S_i and S_j are said to be adjacent if G_i and G_j are adjacent in the hypertree.

□

Notice that this definition of MSBN is the logical consequence of the previous five basic assumptions.

- To finish with this detailed description Xiang just proves that a multi-agent verifying BAs from 1 to 5 can be represented by an MSBN or equivalent. That is, from BAs on a multi-agent reasoning task we got the requirements needed on the representation formalism.

4.2.2 Compilation and inference in MSBNs

In contrast to the single-agent paradigm in Bayesian networks where the reasoning is a process completely centralised, in MSBNs inference needs to be done in an independent way for every agent. Anyhow, this independence, that comes from the nature of the system, can not be total, and then a collaborative method is also a requirement. Xiang's work has been directed to develop a computational framework to allow such a multi-agent uncertain reasoning. The basic assumptions previously described were set to facilitate this collaborative inference. Thanks to the way the agents (DAGs) are structured in the hypertree it is possible to establish *communication* among the agents necessary for an inference process where all agents need to cooperate with their own information.

In [131] there is a thorough description on how inference could be done for an MSBN. In that work, there exist six different types of dlink[2] and the transformation form an MSBN to a Linked Junction Forest (the corresponding entity to a Junction Tree for Bayesian networks) is explained. But further work, summarised in [133] redefines this

[2]D-link is a link between two d-sepnodes.

compilation process in order to simplify the process. It is this latter approach the one that we will review below.

As in Bayesian networks probabilities need to be represented and inferred. Xiang proves that given a multi-agent system that observes BAs from 1 to 5, it is then structured as a hypertree MSDAG. Therefore, the dependence structure of this multi-agent system is a connected DAG G. This assures that a joint probability distribution over the set of variables V ($G = (V, E)$) can be defined by specifying a local distribution for each node and applying the chain rule. That was considered to set out the BA 5. From this last BA and from the definition of an MSBN (def. 27), we could compute the joint probability distribution as indicated in the following example.

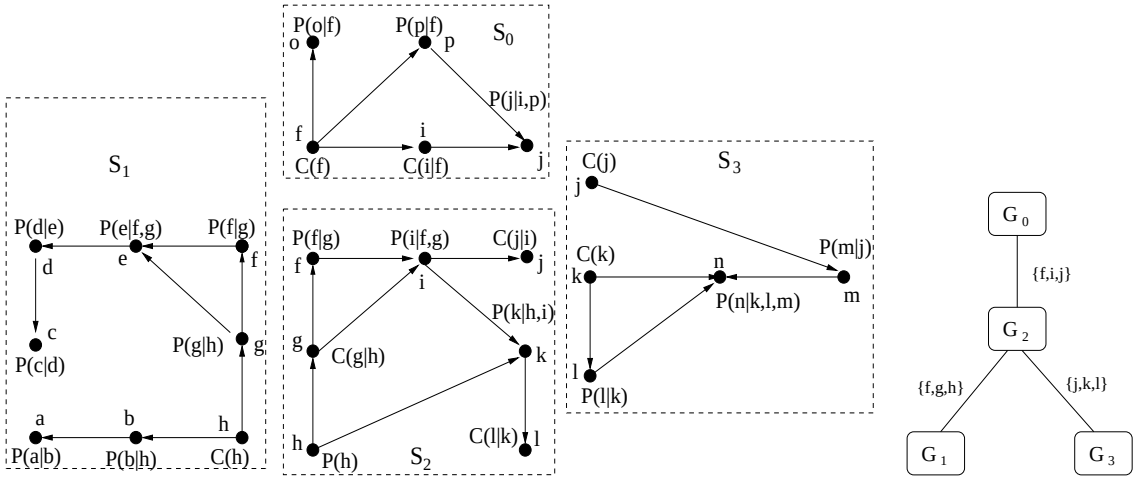


Figure 4.7: An example of MSBN with probabilities indicated.

Example 4 Considering the MSBN in figure 4.6 we now have to add the probabilistic information as 4.7 illustrates. Each $C()$ denotes a uniform potential. To facilitate reading, subindexes have been omitted, that is, in the figure it is written $P()$ instead of $P_i()$ and $C()$ instead of $C_i()$. For subnet $S_0 = (V_0, G_0, P_0)$, we have $V_0 = \{f, i, j, o, p\}$ and $P_0(V_0) = P(j|i,p)P(p|f)C(i|f)P(o|f)C(f)$. Note that $P(p|f)$ and $P(o|f)$ are specified by the vendor of A_0[3], whereas $P(j|i,p)$ could be the result of combining $P_0(j|i,p)$ from A_0, $P_2(j|i)$ from A_2 and $P_3(j)$ from A_3.

So, the joint probability distribution for this whole MSBN is

$$P(V) \;=\; P_0(V_0) \cdot P_1(V_1) \cdot P_2(V_2) \cdot P_3(V_3)$$

[3]Since we are supposed to use a logical circuit example, every agent A_i associated to a graph G_i is supposed to correspond with a certain vendor.

□

This previous formula is equivalent to the JPD obtained in a single-agent Bayesian network (remember Equation 1.2).

It is also remarked the possibility of a shared node x to co-exist in multiple local DAGs also containing its parents, $pa(x)$. For example in figure 4.7 $pa(f) = \{g\}$, $pa(g) = \{h\}$ and $pa(h) = \emptyset$. Both graphs G_1 and G_2 contain each of these parent sets. In these particular cases the occurrence of the shared node x to which $P(x|pa(x))$ is assigned can be determined arbitrarily, since this assignment will no make any difference to the resultant JPD. In the previous figure, the occurrence of f – $P(f|g)$ – is assigned in G_1, the occurrence of g – $P(g|h)$ – is assigned in G_1 and the occurrence of h – $P(h)$ – is assigned in G_2.

It is assumed that for every variable x internal to a subgraph, $P(x|pa(x))$ is specified by the corresponding agent and for each shared variable x, $P(x|pa(x))$ is specified by combining beliefs from all agents involved. Then, BA 5 requires that the joint probability distribution of the universe $P'(V)$ satisfies

$$P'(V) = \prod_{v \in V} P(v|pa(v))$$

being $pa(x)$ the parents of a node x in the total graph G. According to def. 27

$$P(V) = \prod_{i} \prod_{v \in V_i} f(v|pa_i(v))$$

where:

- $pa_i(x)$ denotes the parents of x in the graph G_i
- $f(v|pa_i(v)) = P(v|pa_i(v))$ if $pa_i(v) = pa(v)$ and $P(x|pa(x))$ is assigned to the occurrence of x in G_i, and $f(v|pa(v))$, is a uniform potential otherwise.

Since a one-to-one mapping exists between terms $P(v|pa(v))$ in $P'(V)$ non-uniform terms in $P(V)$, and the uniform potentials do not affect the value of $P(V)$, we have $P'(V) = P(V)$.

Linked Junction Forests

As we already know it is possible to convert a multiply connected Bayesian network (BN) into a junction tree (JT) to perform belief updating by message passing (chapter 1). So, the same would be desirable for its multi-agent extension. Since, each subnet

in an MSBN is also multiply connected in general, a similar compilation method has been developed to perform belief updating in an MSBN by message passing. In fact, all the previous BAs were directed to achieve this objective.

The alternative dependence structure to the single-agent Join Tree (from a specific BN) is, as already commented, the so-called Linked Junction Forest (LJF). The way of its construction is inspired in the JT building. This LJF building from an MSBN is then somehow parallel to the compilation of a BN. We will refer to it as *distributive* compilation, because it will imply distributing each agent shared information to the rest. Figure 4.8 illustrates this relation. Hence, *distributive* compilation foundations imitate traditional compilation.

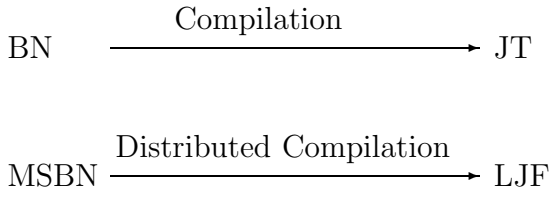


Figure 4.8: Parallelism between compilating BNs and MSBNs.

Yet this variation needs a more complex solution, since we are dealing with a more complex problem. The new compilation procedure needs to include new features and a non trivial adaptation of the algorithms to the distributed scope has to be done. As an initial approach, see figure 4.9 where the main three steps of distributed compilation are enumerated and also connected with the corresponding steps in a BN compilation.

step	compilation	*distributive* compilation
1.	Graph moralisation (G_m)	Cooperative Moralisation
2.	Triangulation of G_m	Cooperative Triangulation
3.	Join Tree construction	Linked Junction Forest construction

Figure 4.9: Compilation vs Distributive compilation

Next, we will make a brief description on how these three basic steps: Cooperative Moralisation, Cooperative Triangulation and LJF construction should be performed. For very interested readers, these processes are explained in detail in book [133]. Here we just summarise them in order to show the more important points for the particular

research of this work. Our greatest interest will of course be the compilation process and especially triangulation, as we have examined the two of them thoroughly in previous chapters. Also, we will see how MSBN could be benefitted from IC mechanism, and moreover, how an alternative way for triangulation [132] can be applied in combination to maximal prime subgraphs decomposition and independent MPS triangulations.

For a single BN the moral graph, the triangulated graph and the JT are all produced in order to preserve the dependencies originally in the network also in the JT (each of them is an I-map). A similar aim is pursued in MSBNs: its graphical structure (MSDAG) is a graph that will also be moralised, triangulated and will give rise to a LJF. The difficulty now is that each agent only has access to its local subnet and none of them *perceives* the entire MSB network. So, cooperation among them is necessary for both moralisation and triangulation, which are subsequently called cooperative.

1.- Moralise cooperatively/distributely the graph

Cooperative moralisation will be performed in an analogous way to probability propagation schemes. So, in the corresponding hypertree[4] one agent will act as the root, A_{root}. This root agent performs its local moralisation on its own graph (G_{root}), calling next to the adjacent agents in the hypertree. By this call, the recipient agents will also moralise themselves locally and later they will send the caller those moral links that affect uniquely nodes in the shared set (d-sepnodes). This process is repeated until all agents have carried out their own graph's moralisation and all *common* moral links have been communicated through neighbour agents in the hypertree.

Figure 4.10 indicates MSBN status for every step in the cooperative moralisation in an example hypertree whereas figure 4.11.(a) illustrates the final situation for this example and finally 4.11.(b) shows an overview of the entire process indicating the previous notated phases. Thicker lines indicate the moral links that has just been added from the local moralisation in the current phase. Blue lines indicate those moral links added because they have been received from other adjacent agent message.

The cooperative moralisation algorithm performs the moralisation in a distributive way, using two main algorithms: one for collecting moral links (which reminds the downward phase of probability propagation) and the second for distributing them (similar to the upward phase).

It has been proved that this cooperative moralisation produces exactly the same moral graph as is the MSDAG would have been treated as a whole DAG corresponding to a single BN. This proof can be found in bibliography, but we will omit it in

[4]We know that we will always have this hypertree from MSBN definition.

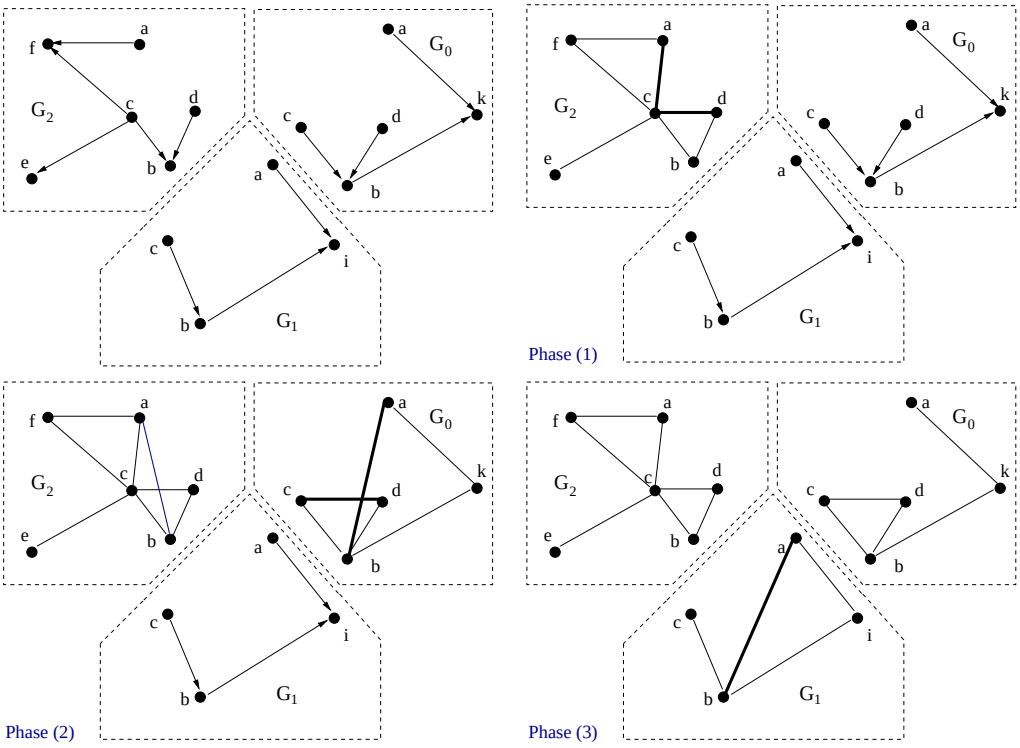


Figure 4.10: Example of applying Cooperative Moralisation .

this chapter to avoid excessive length in this particular look at MSBNs. Again, we should remark that this method is not only valid, obvious as it has been proved, but also attaints the goal of enabling agents' privacy, since only information (moral links) between adjacent nodes (I_i) is revealed.

2.- Triangulate cooperatively/distributely.

Before going further on triangulation, it would be convenable to introduce the concept of *Linkage Tree*. Xiang presents this linkage tree as an alternative representation of the agent interface that will support concise interagent message passing. The need to construct linkage trees will impose additional constraints when the moral graph structure is triangulated into the chordal structure (the step we are explaining now).

To understand what a linkage tree consists of, we could look at it as the construction of a JT from the subgraph projected over the interface node between the two adjacent graph agents we intend to communicate. This reminds how Incremental Compilation constructed partial trees, as we will comment later, but that is not the way it is presented in [133] where other method of obtaining it is developed and justified. We

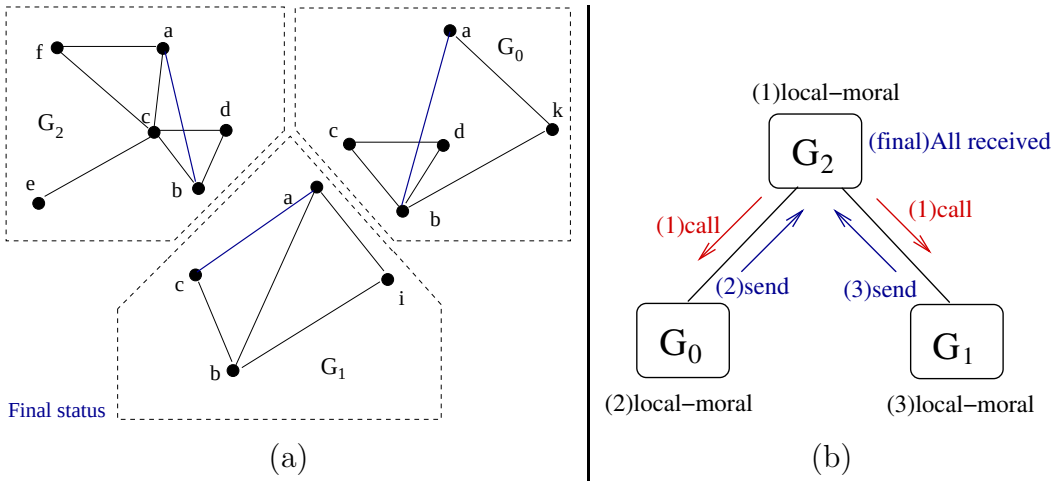


Figure 4.11: (a)Final situation (and phase) of applying Cooperative Moralisation to the previous example. (b)Global overview of the phases on the hypertree.

find that both are equivalent, and the main goal is to introduce the most important elements in a summarised form, but also in our personal form.

An example of linkage tree could be the following that comes from graphs G_0 and G_2 in figure 4.6. In figure 4.12 we can see the full hypertree graph that will be used to illustrate the concept of linkage tree.

In figure 4.13 both local graphs appear already moralised (moral graphs are in this case chordal) and their corresponding local JTs.

And in figure 4.14 the construction of the linkage tree and its situation between the two agents is shown. Every cluster Q in a linkage tree L is called a *linkage* and a cluster in T that contains Q, $Q \subseteq T$ (breaking ties arbitrarily), is called the *linkage host*. This connection is shown graphically with a grey-shaded thick line.

So, the purpose of linkage tree[5] is to manage interagent messages. We should at this point comment on the two kind of messages that are considered in MSBNs framework:

- i-message, where i stands for internal, that is, message internal in an agent.

- e-message, where e stands for external (message), that is, from/to another agent.

As we saw in BA 2, direct communication between agents can only be done on shared variables with an adjacent agent on the hypertreee. Such condition was established for the sake of efficiency, since this will minimise e-messages. There are further

[5]Notice that a linkage tree, as a JT, is composed of clusters and separators.

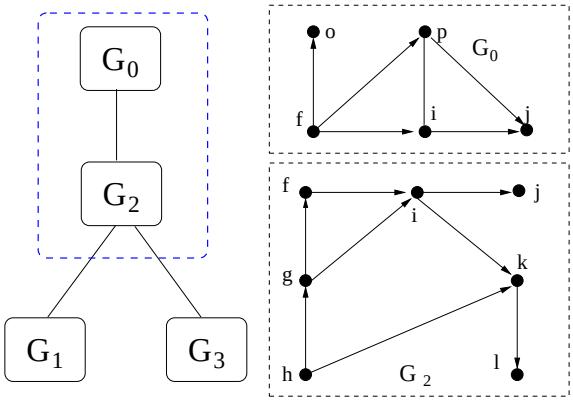


Figure 4.12: The two agent graphs adjacent in the hypertree we will show the linkage tree for. On the left the whole hypertree, the area included on the dashed blue box is the one we are dealing with.

advantages of using this type of communication such as protection of agents' *privacy* and more autonomy for them as well.

To use the linkage tree in a valid way new restrictions in the local triangulation arise. As moralisation, triangulation will be performed in a message propagation scheme. Hence, a root agent will be chosen in order to collect and distribute information, in this case triangulation links or fill-ins. Previously in this work, triangulation task has been described (section 1.5) and also widely studied (chapter 2). We have therefore already seen how a triangulation is generally determined by a certain elimination order of the variables. For cooperative triangulation the elimination order (σ) is so important that there will be specific orders not producing a correct linkage tree.

Then, the main constraint to be accomplished (refer to [133] for details) is the following: given an agent A_i and its corresponding moralised graph G_i, the order σ_i has to satisfy

$$\sigma_i \;=\; (V_i \setminus I_{caller}, I_{caller}) \tag{4.1}$$

where V_i are the variables in G_i, that is $G_i = (V_i, E_i)$, and I_{caller} is the d-sepset interface between this agent and the one that demands the fill-ins from it.

Basically, this means that d-sepnodes to the *upper* agent cannot be removed in the triangulation process until all the rest have been removed. The main drawback of this constraint is the big amount of fill-ins that might be introduced.

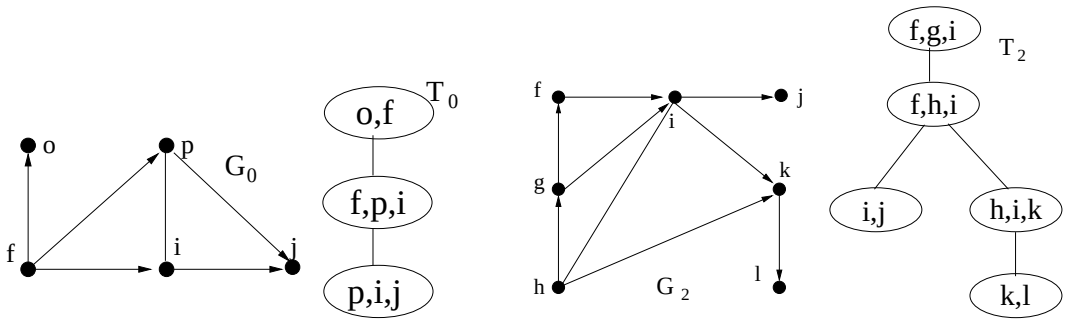


Figure 4.13: Local moral graphs and local JTs for agents A_0 and A_2.

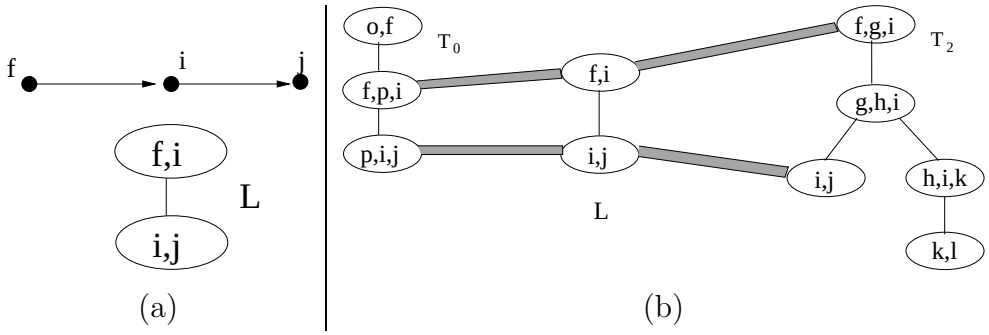


Figure 4.14: (a)Obtaining the linkage tree. (b) Its connection to the two local JTs as an interagent communicator.

The main method sketch (called Depth_First_Eliminate in [133]) is similar to the moralisation, and is written in. As said before, we need to previously determine a root agent that will start the procedure. The first call to Depth_First_Eliminate will go through its adjacent agents and so recursively until reaching a *leaf* agent, A_{leaf}. Then, this/these A_{leaf} will perform a local triangulation subject to the previous restriction (Eq. 4.1). Once A_{leaf} has finished its own triangulation the corresponding fill-ins restricted to I_{caller} will be sent to the adjacent agent A_{caller} that calls to its Depth_First_Eliminate method. A_{caller} will add these received links to afterwards triangulate itself again taking into account the order restriction.

If we follow the previous Collect & Distribute algorithms, we can see that the process is roughly the following one: the root triangulates its graph (G_{root}) once per every adjacent agent A_i subject to the elimination order $\sigma_{ri} = (V_r \setminus I_i, I_i)$. Then it sends the resulting fill-ins (between shared nodes) of the corresponding triangulation to these

agents A_i , and so recursively, returning the fill-ins of G_i restricted to I_i. Once a leaf is reached, A_{leaf} includes the received fill-ins in its graph and it uses a sequence $\sigma_{ic} = (V_{leaf} \backslash I_{caller}, I_{caller})$ to triangulate this resulting own graph, including the new fill-ins and finally sending the corresponding fill-ins to the caller. This process is recursive as well until reaching the root agent again. See figure 4.15 for an example with seven agents. Notice that downwards arrows are the transmitted fill-ins from a caller to its adjacent agent while upwards arrows indicate the returning fill-ins from a called adjacent agent when the function has finished. Finally in order to make the graph "fill-in consistent" all triangulating links are communicated through the tree, and that is what algorithm Safe_Cooperative_Triangulation is charged of, again form the root agent.

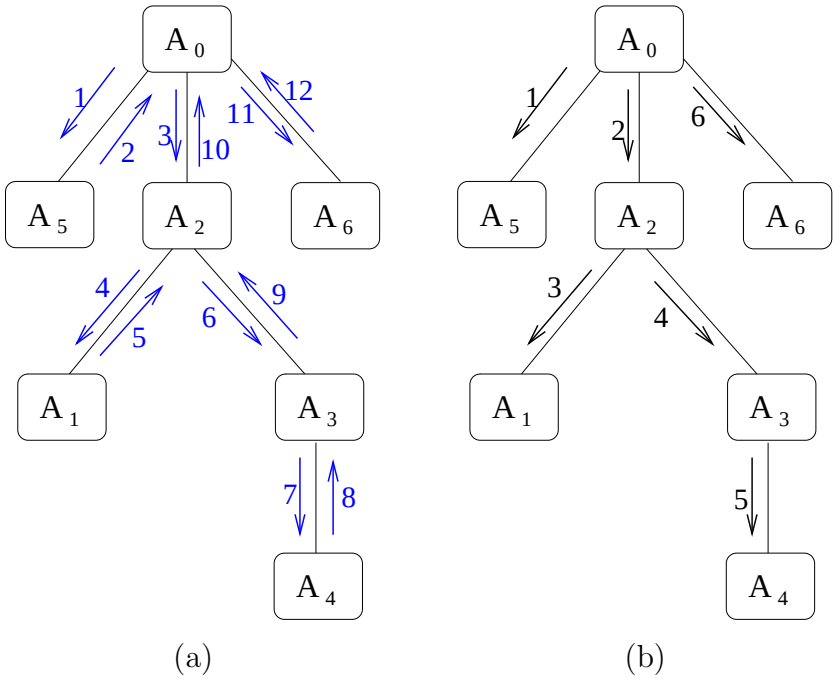


Figure 4.15: (a)An example for the method Depth_First_Eliminate. Arrows indicate fill-ins messages and the number next to them the relative order in the algorithm process. (b)Distribute_Fill-ins, which communicates the produced fill-ins from root to the leaf agents.

Xiang sees that Cooperative_Triangulation does not guarantee that each G_n is eliminable in an order $(V_n \setminus I_i, I_i)$, although this circumstance is quite rare. To ensure completely this requirement, he added a final algorithm to perform triangulation.

The situation that provokes the necessity of applying this algorithm for safe triangulation does not arise often, but it is possible anyway. So, we have to assure that the obtained triangulation is completely valid in all cases. Basically the problem comes from a situation where fill-ins received in a collect phase, which takes place after a certain local triangulation when Depth_First_Elimante, could provoke a change in such a previous triangulation. Let us see an example: an agent A_1 can be adjacent in the hypertree to more than one agent, let us say A_2 and A_3 for example. At some point agent A_1 will locally triangulate its graph G_1 in an elimination order σ_{12} with the restriction $(V_1 \setminus I_2, I_2)$ producing or not a set of fill-ins. But then, in the collecting phase, it can happen that A_1 receives, for example, a fill-in from A_3. And maybe now, with this new link, *unknown* by A_2, since $I_2 \neq I_3$ the order σ_{12} will include new fill-ins that are impossible to be considered if we do not make another iteration, and that is what Safe_Cooperative_Triangulation does.

Finally, in this step it could be interesting to mention that obviously Cooperative Triangulation includes more fill-ins than a global triangulation. That is logical, but it is a price to be paid in order to guarantee privacy and autonomy for every agent. Nevertheless, this amount of *extra* fill-ins is not so big as could have been expected. Experimental studies have shown that cooperative triangulation produces reasonably sparse chordal graphs.

3.- Construct the corresponding Linked Junction Forest.

Once local graphs have been moralised and triangulated, it is possible to construct also local Join Trees. Notice that this reminds the partial trees we obtained in the Incremental Compilation technique. So, local JTs can be obtained in the same way we did for a single BN (chapter 1) and as we introduced previously when explaining the use of Linkage Trees. So, for every agent A_i we will have its graph G_i already moralised and triangulated $G_{i_m}^t$ and through a *typical* compilation process its local JT T_i can be easily reached.

There is another element needed to construct the LJF, and it is the Linkage Tree. There must be a linkage tree $L_{i,j}$ from a T_i to every T_j iff $A_j \in adj(A_i)$. The formation method for this linkage tree was briefly explained in the triangulation step, as well as the way it should be connected to the clusters in the trees T. Here we will not add more detail about this creation process. It is important to remark that the requirement about the elimination order was closely related to this linkage trees, and in fact, this requirement will guarantee that there exists such a linkage tree. Also, again the hypertree structure and a selected agent as A_{root} will mark this step. For example in a

MSDAG structure as the one depicted in figure 4.16, local trees T_0 to T_4 together with the linkage trees $L_{0,2}$, $L_{2,1}$, $L_{2,3}$, $L_{3,4}$ will constitute the LJF structure.

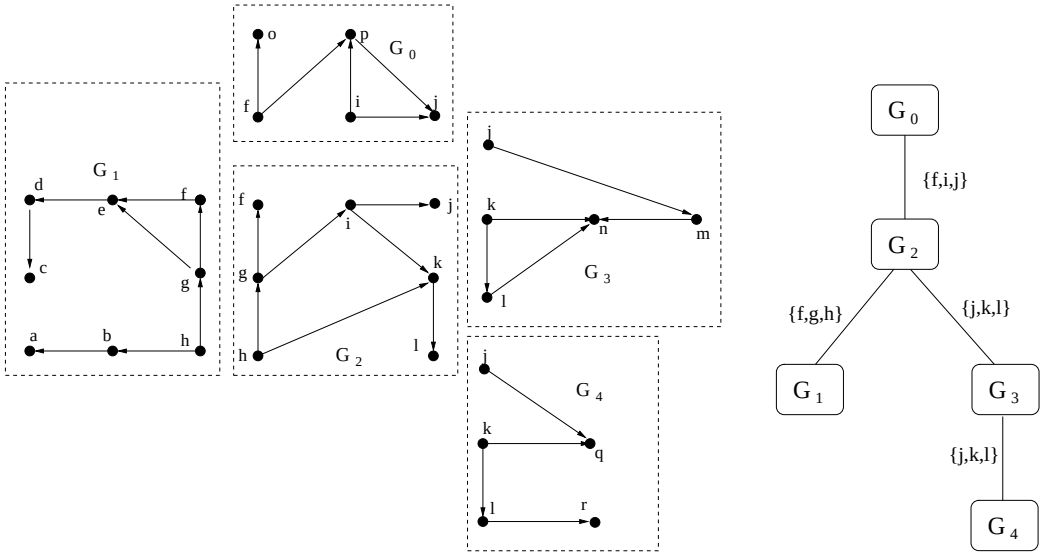


Figure 4.16: Example MSBN to be moralised and its hypertree (right part).

Then, with the local JTs and the linkage trees constructed by each agent, the hypertree MSDAG in the original MSBN has been converted into a different structure, which is termed *linked junction forest* (LJF). That was the purpose of what we called the *Distributive Compilation* (see fig. 4.8). In def. 28 a formal definition of this LJF structure is given.

Definition 28 A linked junction forest F is a tuple (V,G,T,L):

- $V = \cup_i V_i$ is the **total universe** where each V_i is a set of variables called a subdomain.
- $G = \sqcup_i G_i$, where each G_i=(V_i,E_i) is a chordal graph such that there exists a hypertree Ψ over G.
- $T = \{T_i\}$ is a set of JTs, each of which is a corresponding JT of G_i.
- $L = \{L_i\}$ is a collection of linkage tree sets. Each $L_i = \{L_{i,j}\}$ is a set of linkage trees, one for each hyperlink incident to G_i in Φ. Each $L_{i,j}$[6] is a linkage tree of T_i to respect to a hyperlink $V_i \cap V_j$.

□

[6]Notice that $L_{i,j} = L_{j,i}$.

See that the LJF is an alternative dependence structure for a multi-agent system whereas the JT is the local dependence structure of a certain agent A_i. Xiang also proves that this structure represents the dependencies of the original MSBN and permits message passing for inference being LJF also an I-map. For that proof he refers to the concept of graphical h-separation.

Inference in MSBNs

As it happened for BNs, multi-agent inference will manage the probability propagation using potentials (def. 10). They will allow this scheme of belief updating by concise message passing. So, the conditional probability distributions in an MSBN are converted into potentials in the LJF in order to make inference process available. The potentials will not only be assigned to cliques (clusters) and separators in a JT, because LJF contains also linkage trees. These linkage trees will be charged of inter-agent communication, so they will need to register probabilistic information as well. We could distinguish three kinds of potentials then: (1)a potential for each local JT, (2)a potential for each separator in each linkage tree and (3)a potential for each linkage tree. Finally, we could consider a joint system potential (JSP) for the entire LJF representation.

To do this brief explanation, we will use a simple example MSBN whose structure and conditional probabilities $P(x|pa(x))$ are in figure 4.17. If we follow the distributive compilation process that has just been detailed, the corresponding LJF could be that in figure 4.18.

As we saw in chapter 1, initially the CPTs have to been assigned in some clique of the tree and the other potentials remained as unitary (probability trees) or uniform (probability tables), that is, they do not contain information until propagating messages. In LJF some potentials are assigned and the rest are defined (or just say derived) in terms of these potentials.

We are not going to look these into detail, just remark that the procedure of assignment for potentials is similar to the single-agent method. Every CPT has to be reflected in one potential that contains all the affected variables, that is, for variable x, the chosen cluster must contain x and its parents. The rest of the clusters, separators and linkages in the linkage trees will have initially a uniform potential. On the other hand, the derived potentials will be in terms on the previous ones.

There exists an *analogous* concept to the JPD in BNs. This is called the joint system potential, JSP, and it is defined over the universe V associated with the LJF

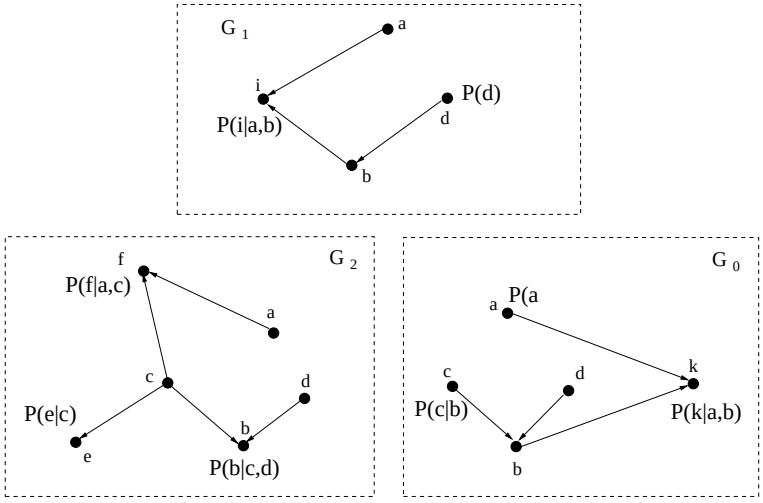


Figure 4.17: Simple MSBN to illustrate the LJF for inference.

F.

We have decided to skip details (probabilistic formulas) about messages passing, since it goes beyond the scope of this revision. We just remark this is a more elaborated version than the single-agent paradigm frameworks. Let us simply remind the difference between i-messages and e-messages. The former are intra-agent and there will be similar to a scheme of probability propagation, the latter are inter-agent and will do the communication the potentials over the shared variables ($I_{i,j}$). In figure 4.19 a global overview of this agent communication is given for the previous example. Blue arrows illustrates the process when the communication is activated: A_0 first, then A_2 and finally A_1. When A_1 is activated, it passes the message to A_2 over their linkage trees. After local processing $A2$ will pass the message over their linkage trees. The black arrows indicate the sequence of the message passing. When the first round (COMMUNICATE_BELIEF) is completed, the second round starts (DISTRIBUTE_BELIEF). After local processing, A_0 passes the message back to A_1 (following blue arrows). In [133] it is shown that after these two rounds of interagent passing, all agents' beliefs are updated correctly.

Then, this method could be finally summarised with a familiar scheme: Belief_Collection + Belief_Distribution, where the two methods (phases) are more complex than previous. They include new operations such as absorption through linkages, unification and updating of beliefs.

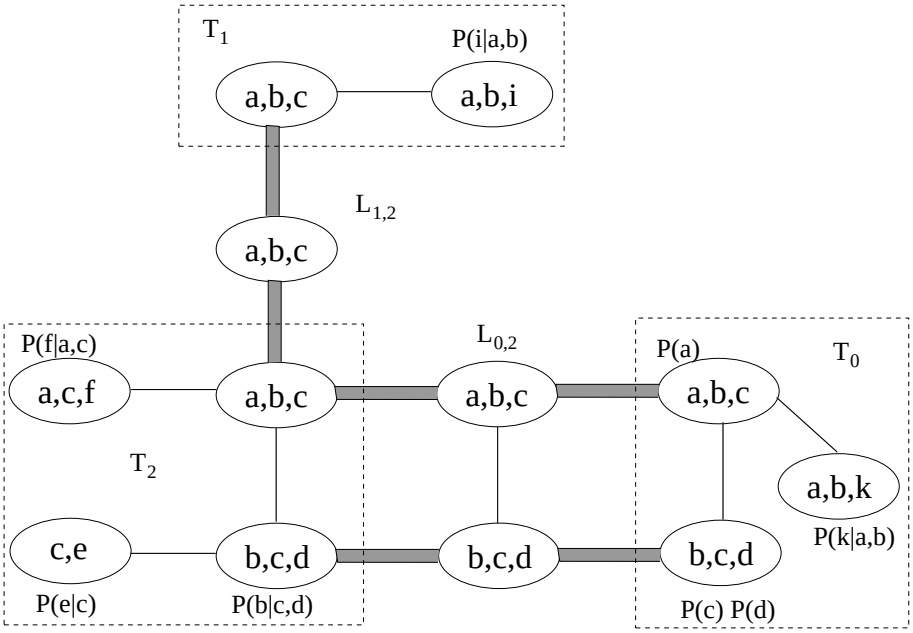


Figure 4.18: Linked Junction Forest for the MSBN in figure 4.17.

It is clear that this inference process is more complex than the single-agent procedure with JTs (chapter 1), even if it is based somehow on it.

From this inference framework, some relevant points can be extracted. First, there is a well studied and defined proposal to perform inference in MSBNs, and that will be used for other approaches as OOBNs (see next section). And second, and even more important for this whole work, triangulation becomes again in a decisive point since the complexity[7] of the triangulation carried out is going to influence how complex the inference process results. And here the inference gets even more complicated. The reason for that statement is that both local JTs T_i and the inter-agent linkage trees are constructed from that triangulation process. And since the LJF is the structure used for inference and it is made up of these two kind of trees, the effect of triangulation on inference is clear. Although the proposed method is correct, this commented fact has favoured the search of other alternative methods to optimise this *distributive* triangulation. If *normal* (single-agent) inference could in occasions be quite hard to perform, multi-agent inference can make some efficiency problems even more serious.

[7]This can be seen as the number of fill-ins and more precisely as the total (sum) size of the formed clusters.

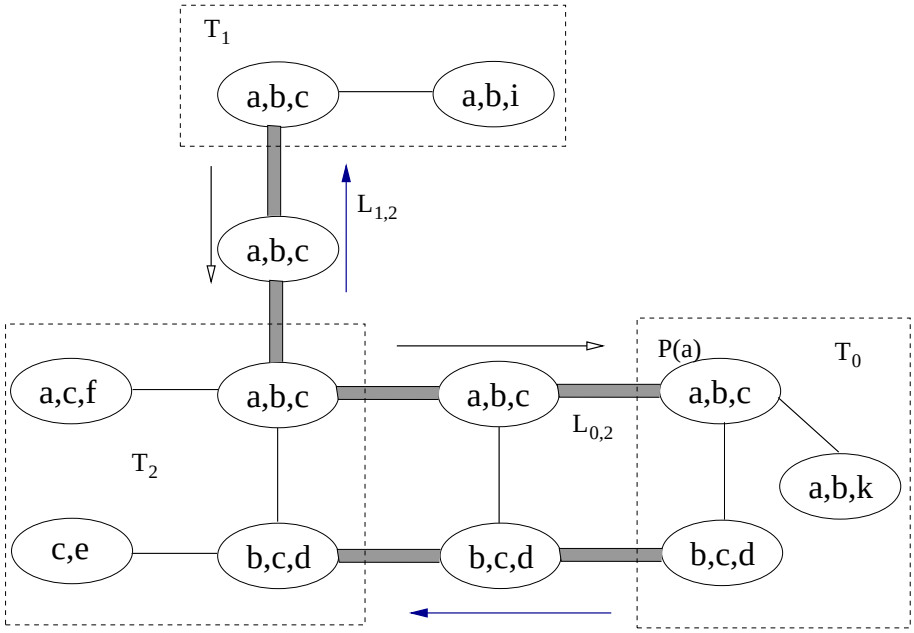


Figure 4.19: Communication among agents for MSBN in figure 4.17.

The work [132] presents a quite attractive fill-in propagation scheme for inference in MSBNs, that will be reviewed in subsection 5.3. The main interest of this new idea is its double relation with MSBNs and OOBNs, together with a third connection to IC we will try to justify at the end of the current chapter.

4.3 Object Oriented Bayesian networks

To start with, we could say that Object Oriented Bayesian networks try to put into practice the ideas of the Object Oriented Approach in the field of BNs specification. This general approach has achieved great significance in the last decades in many situations. For that, several works ([66, 72, 5]) has been directed to obtain an Object Oriented Specification of BNs with the purpose of incorporating most of the *modular* but cooperating features this Object Orientes philosophy offers. We are going to focus on the approach in [4], whose specification is more recent. For further knowledge of this approach, we refer to [7] where apart from the foundations, other detailed processes and applications of OOBNs can be found.

The commercial tool HUGIN has added in the last releases the OOBN functionality,

based on this specific approach, and providing a nice graphical interface to model this kind of networks.

As we did with the MSBN a description of this approach will be presented below.

4.3.1 Basics on this OOBN framework

The main purposes of [4] were:

1. Top-down methodology for construction.
2. Easy-to-use tool for constructing large BNs.
3. Encapsulation and hierarchy as in OO.

With this new OO Specification the modelling task is simplified, but also we get a very compact specification of knowledge that can be really exploited. Figure 4.20 is an example of this approach. In this sample OOBN, we can find familiar and BN-related elements such as variables (nodes) and directed links (relations). It is evident that OOBNs are Bayesian networks (fragments), but other new elements and relationships appear in order to manage the OO features: double-line links, shaded nodes, dashed nodes, boxes, ... represent extended definitions that we will next review.

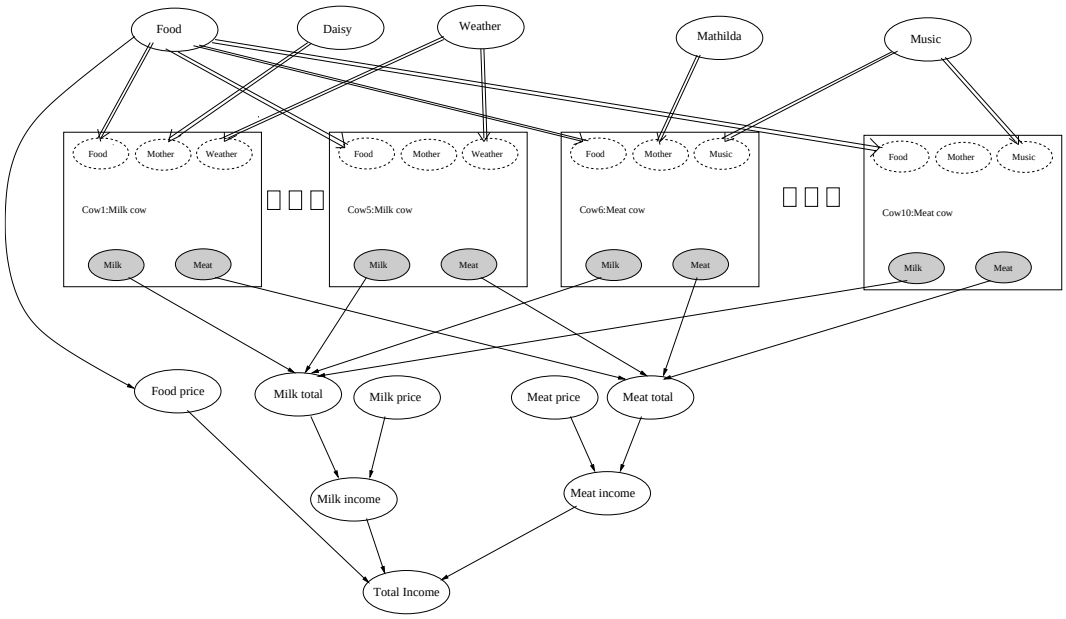


Figure 4.20: Example of OOBN

In this framework the concept of *class* is used. A class will be a description of a set of objects with the same structure, behaviour and attributes. An object will present

its own identity, status and behaviour. One class encapsulates nodes and restricts the scope of its nodes to the interior. Then, a node inside a class can not be seen from outside and also this node will not see that information external to its containing class.

The basic element in this object oriented framework is effectively the object. We should look at an object as a class instantiation, i.e. the class is a kind of specification and instances of the class (objects) are being created along they are needed for the system. There will always be an outermost class denoted as encapsulating class, since from the current scope, there is no other class outer than this. This *special* class is viewed as both a class and instance.

It is possible that a class has instances of other classes. For example, in figure 4.20 there are 10 instances (objects) of the class Cow. Also, in this figure, we observe links between nodes in the OOBN and nodes inside Cow objects. This interaction is indispensable, and the OOBN framework defines how it should be done. Basically, two circumstances are permitted for class interaction: (1) a node inside one class could have parents outside and (2) a node outside a class can have parents inside this class.

Definition 29 A **class** is a Bayesian network fragment defined by a triplet of nodes set $(\mathcal{O}, \mathcal{I}, \mathcal{P})$ where

- $\mathcal{O}$ is the set of output nodes, which can be parents of other nodes outside instances of this class.
- $\mathcal{I}$ is the set of input nodes. These nodes are not in the class, but are used as parents of nodes inside instances of this class. Input nodes can not have parents in the class.
- $\mathcal{P}$ the set of protected nodes (or private nodes), which can only have parents and children inside the class.

And $\mathcal{O}$, $\mathcal{I}$ and $\mathcal{P}$ must be three disjoint sets.

These nodes will be related by links of two main types: (1)*regular links*: directed links as those in BNs and (2)*reference links*: special kind of links that will correspond an input node to some *external* one. □

In graphical representation input nodes are dashed-line limited and output nodes are shaded. Protected nodes do not have any special representation, since they are the *traditional* nodes.

The input nodes of a class could be understood as its input parameters. So, what an instance needs in order to be created is values to these input nodes. That is also necessary in order to specify the CPTs, and we need to specify the parents of a node

during probability specification, that is, class specification, not during instantiation (to instantiate a class we need first its specification). For that, no node inside a class can have parents from outside. But that also provokes the introduction of a new type of node, needed for this *assignment* of input nodes: the reference node.

A reference node will point to a node (in this case referenced) in another scope (class). It is similar to a pointer, but in fact it maintains its proper characteristics and it is somehow a copy of the node it refers to. The states and the potential have to be the same as its referenced node. And the link that connects a reference node to its corresponding referenced node is named reference link.

In the previous example, let us focus on instance Cow6 (see fig. 4.21). In the specification of cow, there are three input nodes (Food, Mother and Music) and two output nodes.[8] If we look at the input node Mother (reference node) there is another external node pointed to it (referenced node), *Mathilda*. Sometimes double-line is used to represent graphically a reference link. So, this means that the node Mother in the instance Cow6 has been bound to *Mathilda*. The most important implications of that is that all possible children that Mother had in the specification of Cow will be children of *Mathilda*. Notice that arrows have the direction "referenced node → reference node".

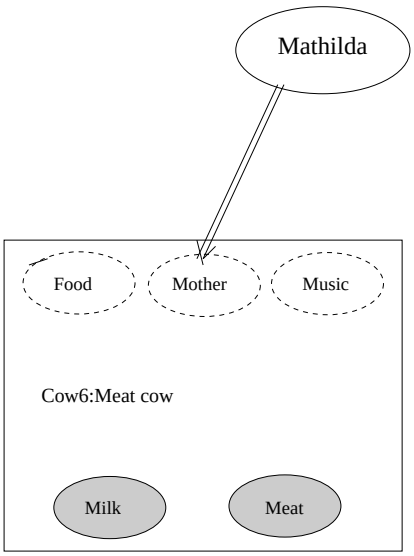


Figure 4.21: Example of reference node and reference link (double line).

[8]Note that the rest of nodes do not appear, since this specification is of a more external class, and then these protected nodes are hidden or unknown for it.

To simplify specification, it is also allowed that a reference node presents a *default potential* used to create a regular node with the same states as the reference node when no referenced node has been specified.

All input nodes are reference nodes. But this is not uniquely reserved to them, extending the use of reference links and reference nodes we can get other functionalities. For instance, to enlarge the scope of a node in an instance to outside the encapsulating class output nodes could be reference nodes[9]. Every node, except input nodes, of a class can be marked as an output node, becoming them visible from outside instances of a class. And the interface of a certain class is the union of input and output nodes ($\mathcal{I} \cup \mathcal{O}$).

A class is then a fragment of BN (nodes and directed links) which may contain special nodes (instances and reference nodes) and special links (reference links). Clearly this is an extension of the definition of BN, a class with no instances, no reference nodes and empty input and output sets will be a normal BN.

One structure of great utility for OOBNs is the Instance Tree (IT). Because when constructing OOBNs instances of other classes are inside a unique encapsulating class, this tree of instances can always be built. This method is shown in alg. 22.

Algorithm 22 Constructs an IT from an OOBN.

```
1: function GET_INSTANCE_TREE(Class C)
   ▷ C is a class, and then a Bayesian network fragment.
2:     IT ← ∅
3:     tn ← C.getEncapsulatingClass()
4:     IT.addTreeNode(tn)
5:     for all Instance node IN_i ∈ C do
6:         IT.newChildBranch(GET_INSTANCE_TREE(IN_i))
7:     end for
8:     Return IT
9: end function
```

An easy example of this procedure can be seen in figure 4.22.

As we can see this instance tree will have the encapsulating class as the root, and each instance inside this class will define a subtree with the instance as the root, and so on. In figure 4.22, the instance B for example will produce the subtree $J1 \leftarrow H1 \leftarrow B \rightarrow H2 \rightarrow J2$, where the root is B.

[9]Graphically a reference output node will be shaded and will be draw in a dashed line.

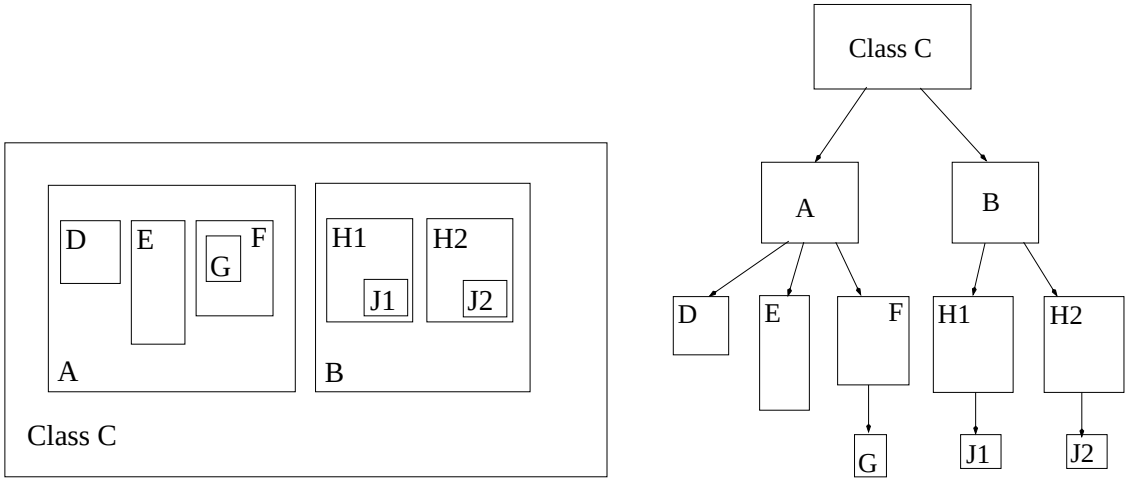


Figure 4.22: Example for constructing and Instance Tree. Left: the class specification. The corresponding IT.

This instance tree will play a role for communication among classes, as section 5.4 will indicate. See that interface nodes of an instance are the only nodes that allow links to upper nodes in the instance tree, since only input/output nodes are allowed to interact with other nodes in the encapsulating class. Then, all communication between nodes will be through these interface nodes in such a way that if two instances which are not adjacent (neighbours) in this IT intend to communicate, the communication should follow the path in the tree between them. A more real example where a class is specified with its nodes and links is shown in figure 4.23.

Apart from the already commented links: reference and *regular* links, there is a possible third kind called construction links. These are useful for the construction or modelling of an OOBN. Since this task goes beyond the scope of this work, we just comment their existence.

In order to assure that the constructed OOBN is not *illegal*, i.e., is a valid OOBN, there are some restrictions over the reference links. Basically, these restrictions try to avoid cycles in the network and other structural problems for the IT, since referencing can finally provoke regular links that are not so immediate. Including the so-called *reference tree restriction* (see def. 30) these cycles are avoided.

Definition 30 Let us call **reference tree** a tree whose root is a referenced node and whose branches are formed by all the nodes that reference this root node, and so. □

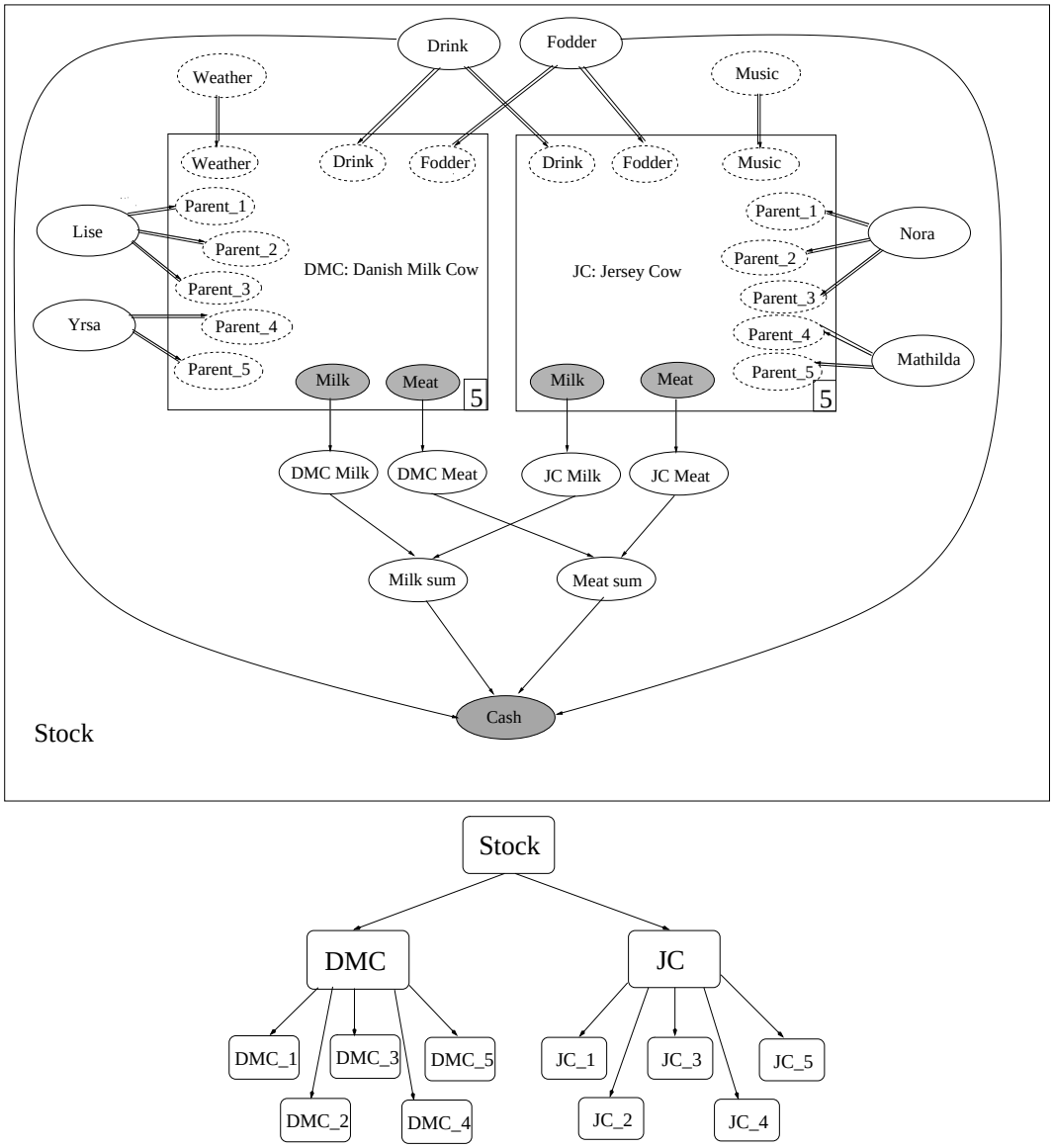


Figure 4.23: Another example for constructing and Instance Tree taken from [7]. The number X in a small square box on the right bottom corner of an instance means that there are X instances of that class. It is for simplicity in visualisation. Notice that Parent_i means the input node *Parent* for instance i, the same happens with the instances DMC_i and JC_i in the instance tree.

Then the **reference tree restriction** imposes that reference links cannot be specified between two simple nodes in the same class, and no node can be the referenced node

for more than one reference node in the same instance.

The implications of this restriction can be listed as:

1. Input nodes cannot be used as reference nodes for simple nodes specified in the same class but only for input nodes of (other) instances. Two input nodes in an instance cannot have the same referenced node.

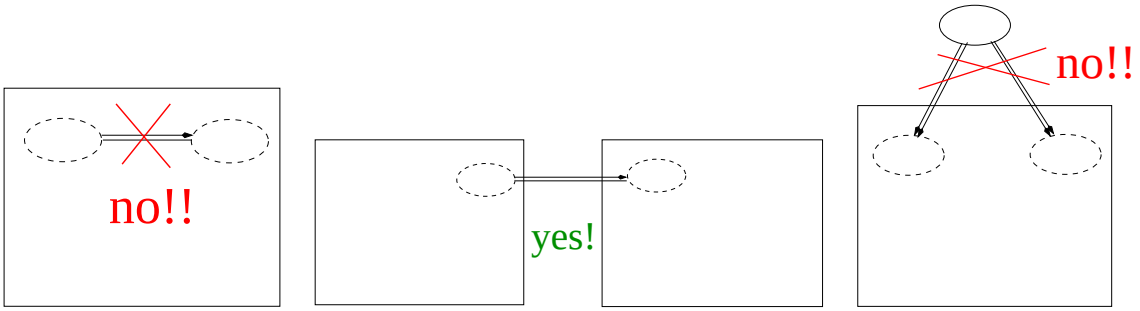


Figure 4.24: First implication: valid and invalid situations

2. Output nodes can be used as referenced nodes for output nodes in the encapsulating class and for input nodes in a different instantiation in the encapsulating class. Two output nodes of a class can not have the same reference node.

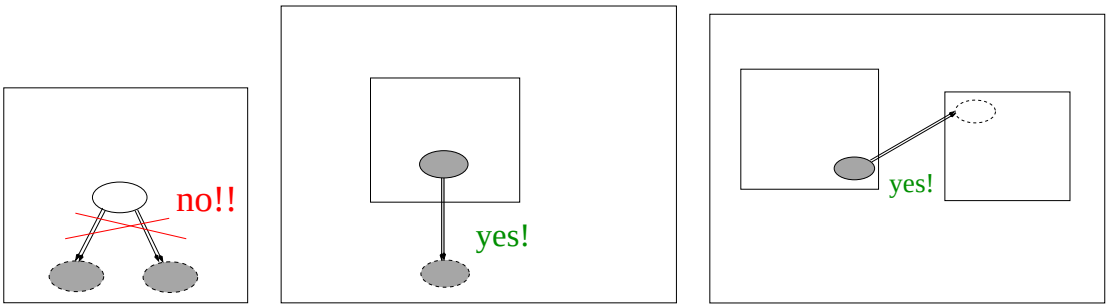


Figure 4.25: Second implication: valid and invalid situations

3. Protected nodes can be used as referenced nodes for input nodes of other instances only. Protected nodes will always be referenced roots if they occur in a reference tree, as they can never be reference nodes.

4. A chain of reference links can go in both directions (further inside instances or further out) but once it begins going inside, it cannot go out again.

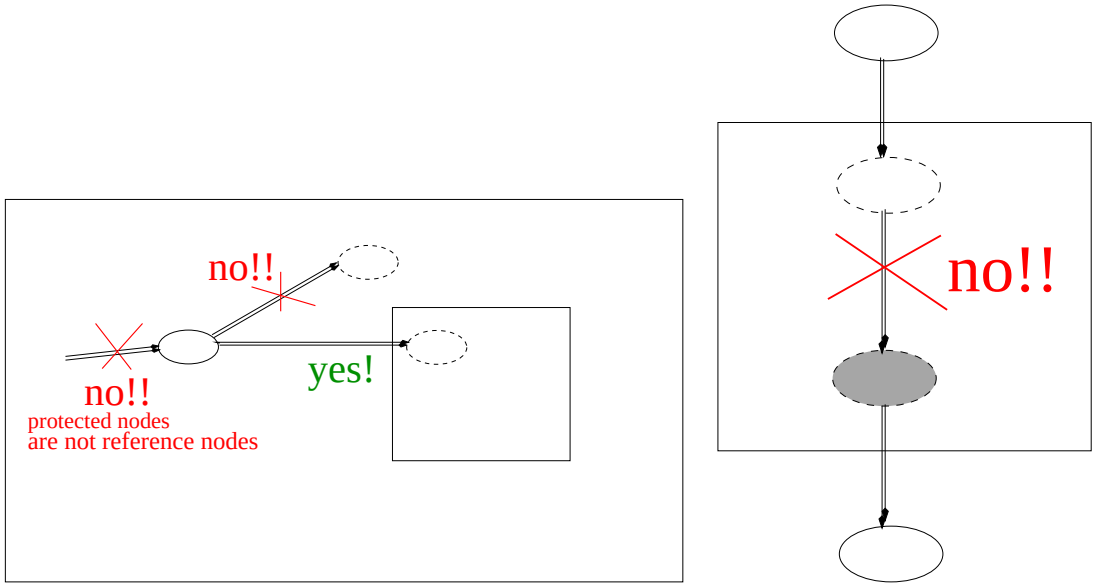


Figure 4.26: (a)Third implication: valid and invalid situations. (b)Fourth implication: what cannot happen with the reference tree.

It has also been proved that, despite this reference tree restriction, OOBNs do not lose expressiveness, because there is always a valid alternative way to express whatever needed without the previous prohibited operations.

It is important to recall the Instance Tree at this point and be aware of the following theorem [7]: "All nodes in an instance are independent of nodes in instances higher up in the IT given its interface".

So, we could also say that the interface d-separates an instance from those instances containing it.

There are many other issues on OOBNs that are necessary to know properly the framework, but which do not affect so directly to the triangulation and inference tasks. So, here we will just go quickly over them.

Firstly typing, that is, the kind and set of states for a node has to be defined and checked to avoid errors. Also, one very nice utility of this OOBN framework is the ability of working with dynamic domains, and also with repetitive structures. For the dynamic environment, they have designed a *time slice* class that could be instanced in different temporal moments. This particular usage has also introduced a new kind of link, the *temporal dependence link*.

Another point to analyse is the class hierarchies. This is another object oriented feature which can be quite interesting for model construction and the current framework takes advantage of it. So, thanks to this hierarchy it is possible to simplify the specification of similar classes, and also to organise knowledge in a hierarchical (and normally better organised) way. But for us, it will be specially interesting because it may allow automatic updates in classes sharing some properties. This quick updating presents a certain relationship with the Incremental Compilation method and is one of the advantages Plug & Play OOBNs have. Details about how hierarchy is defined in OOBNs will be omitted. We just indicate that a subclass S from a certain class C, must have at least the input, protected and output nodes of C, although S can present other extra ones. It is similar to class inheritance in an OO programming language. And the links in the structure if the subclass S could not be the same as C (as inherited procedures in a subclass can be different too). See how the *Danish Milk Cow* and the *Jersey Cow* in figure 4.23 could be sub-classes from a same class, for example, the one depicted in figure 4.27

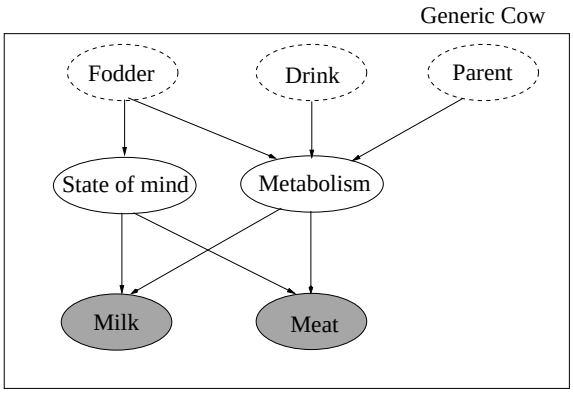


Figure 4.27: One possible specification for the class Generic cow, that could be a superclass for Danish Milk Cow and Jersey Cow.

Chapter 5

Combining the Incremental Compilation concept with *modular* Bayesian structures

By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest.

Confucius. (551–479 BC)
Chinese philosopher and reformer.

5.1 Introduction

In the previous chapter, basic notions for Bayesian modular structures in general, and for MSBNs and OOBNs in particular has been done. They will be needed to present the issue treated in this chapter, where we will deal with inference in OOBNs.

Initially, OOBNs were presented as a fantastic tool for network modelling and construction in large and complex domains. The possibility of dividing a big domain in subdomains, their suitability for dynamic representation and repetitive structures among other features make them a nice extension of BNs for systems with those requirements.

However, since they were developed mainly for this modelling purpose, when reasoning and inference was necessary over an OOBN, no specific methods for inference had been defined. Normally, the way to perform inference consisted of building the equivalent Bayesian network, and then perform inference in the *regular* way. That was

perfectly valid, but it provoked some drawbacks:

- The philosophy of the OO framework is then lost. To transform an OOBN into the corresponding BN, all protected nodes become visible, reference nodes and links disappear making the corresponding *substitutions* and input/output nodes do not have a special meaning anymore.
- Then, the capability of being easily used in dynamic and changing domains, or with repetitive structures is also lost.
- And finally, since OOBNs were thought to manage big and complex domains, the resultant BN can bee too big.

Thus, even if this inference method is perfectly correct and maybe needed for certain situations, it seems quite desirable to have an inference procedure that allows us to work with the OO paradigm. And this was the reason to design other alternative mechanisms without failing to keep the original philosophy. As explained in [7] two other inference ways are so far possible: building an equivalent MSBN (subsection 5.2) or triangulate the instance tree (section 5.4).

5.2 The link between MSBNs – OOBNs

As we were describing the MSBN structure and also the OOBN approach we could just find some common points. Let us return to some of the purposes that MSBN tried to reach:

1. Capability of managing large and complex domains.
2. Modularity, in order to be able to treat some components independently.
3. Hidden parts, that is, some information that is private for the different *agents*.
4. And of course, some kind of communication between these agents to get a cooperating system.

It is clear that all the enumerated points remind an Object Oriented Approach. Fortunately, this approach already exists, and we have just presented the main ideas in the previous section. But then, these OOBNs presented one drawback when we wanted to compute with them. Before, they had been quite useful for modelling systems.

Nevertheless, when propagating they are just translated into a whole BN. Doing this transformation means losing all the power of the OO utility. The process of inference has been deeply studied for the MSBNs. Since the similarity MSBN-OOBN is obvious, what had to be proved is that there exists a way of making a OOBN correspond to an MSBN system. If this correspondence is possible, we will maintain the modular structure of the OO approach. We could take advantage of the easy use of OOBNs and of the well defined process over MSBN.

This structural[1] *equivalence* is possible. In [4] a first method to translate an OOBN to its corresponding MSBN is roughly outlined (see alg. 23).

An MSBN (set of Bayesian Network Fragments (BNFs)), representing the OOBN is obtainable from the specification using the recursive algorithm shown in alg. 23, where **T** is the class being compiled into an MSBN:

Algorithm 23 Obtains the corresponding MSBN to a given OOBN.

```
1: procedure TRANSLATE(OOBN O,MSBN M)
2:     T ← O.encapsulatingClass()                          ▷ T is a class
3:     M ← ∅
4:     for all Instantiation Inst_i ∈ T do
5:         BNF t ← T.getFragment(Inst_i.simple_nodes + Inst_i.linked_output_nodes)
       ▷ t is a BNF, that is, a Bayesian Network Fragment.
       ▷ Note that if an output node is referenced by an input node of another instantiation in T it is
       also defined as being linked.
       ▷ getFragment(set_of_nodes) takes obviously the corresponding links as well.
6:         M.addSubnet(t)
7:         Inst_E ← t.getEncapsulatingInstantiation()
       ▷ The encapsulating instantiation will be T if t is the encapsulating class.
8:         t' ← M.getBNF(Inst_E)
9:         Hyperlink HL ← {t → t'}
10:        M.addHyperLink(HL)
11:    end for
12: end procedure
```

Two things have to be considered in order to understand the previous algorithm:

- Each BNF represents the scope of a class. So we should add the reference linked

[1] Let us remind that OOBN and MSBN approaches can share some points, but their scenario and nature are much different.

input or output nodes of each instantiation, but in fact, these reference linked nodes are represented by the referenced nodes (for input nodes) or by the reference nodes (for output nodes). That is because these will be the shared nodes (from an MSBN point of view) with the upper (containing or encapsulating) class.

- If two BNFs have a hyperlink, then one of the instantiations they represent is encapsulated by the other. The d-sepset between two such BNFs is the nodes they have in common which can only be nodes from the interface of the encapsulated instantiation.

In [4] it is justified that the obtained MSBN is correct. First, the set of subnets (accumulated by statement in line 5) have no cycles, since there are none inside a class specification, and this accumulation is made class after class. Secondly, the formed structure, determined by the hyperlinks added by each statement of line 8, is effectively a hypertree. This is sure because every fragment is created by an instantiation and every instantiation can uniquely be inside one certain class (or instantiations of a class). Finally, they also proved why interface between two subnets are d-sepsets (omitted here).

On the other hand, it can be seen how an MSBN could give rise to a valid OOBN. We could construct a valid OOBN from any given MSBN following the inverse procedure to the one shown in alg 23. The idea is that every subnet could be a class and the interface nodes the input/output nodes of the class, depending on the kind (direction) of links they receive. Besides, if in an MSBN we always guarantee the existence of the corresponding MSDAG which is an hypertree, this can be like the *equivalent* element to the Instance Tree. See how in both cases interface renders an agent/class independent from the rest of the tree (agents' hypertree or instance tree) in a down-top direction.

So, a method for doing the translation $OOBN \rightarrow MSBN$ is valid, and it is possible to find a way of making a certain $MSBN$ be represented by an equivalent $OOBN$ ($OOBN \rightarrow MSBN$).

5.3 A fill-in propagation scheme for inference in MSBNs

As indicated before, apart from the message passing scheme on point 4.3.1, some other alternative inference mechanisms have been developed to perform inference in

MSBNs. A recent work [134] makes a revision on some of them, where some results are said to be of relevance to improve certain single-agent oriented inference methods.

Among all these new methods, one specially appealing work for our particular purposes is [132]. Since the original message passing method for inference can be seen as an extension of HUGIN inference, in this work they seek some manner of reducing space complexity. They succeed in obtaining an adaptation of Shenoy-Shafer and also Lazy Propagation [81] for the multi-agent paradigm. This *lazy* way of proceeding seems more suitable to modelling and inference in very large domains which also maintains the flexibility for constructing MSBNs.

Lazy Propagation

We have already studied Shenoy-Shafer propagation. Let us give a basic notion on lazy propagation. The underlying philosophy in this *lazy* framework is to retard any mathematical operation (reduces computation by avoiding unnecessary combinations) until it is really necessary. As Shenoy-Safer, Lazy Propagation is static. Each cluster C in the junction tree will hold the assigned distributions, but as a set (combinations will be done when needed). The belief table of a cluster C is defined the same as SS, but the product is not explicitly computed. And finally, each message sent over a separator is a set of tables which is over a subset of the variables in this separator. Its efficiency improvements are then obtained by maintaining a multiplicative decomposition of clique and separator potentials.

Figure 5.1 illustrates an example of a Bayesian network already moralised (undirected lines indicate moral links) and chordal (fill-ins are indicated by dashed lines). On the right, one possible corresponding join tree for this triangulation. Next to a clique are the potentials associated (prior probability distributions), but in contrast to SS architecture, this is indicated as a list (set) of potentials. Remind that SS combined them into a unique potential. Then, if for example the message from C_{DEF} to C_{BCD} is sent, in Shenoy Shafer this message would be $\{\sum_{E,F} P(E|D), P(F|D)\}$ with complexity 8 whereas in Lazy the sent message would be $\{\sum_{E} P(E|D), \sum_{F} P(F|D)\}$.

Fill-ins Propagation

As seen in subsection 4.2, the linked junction forest (LJF) method compiles the subnet at each agent into a junction tree (JT) that we denoted T_i. Messages between two adjacent agents A_i and A_j are passed through the linkage tree $L_{i,j}$, being this structure the only element to communicate this pair of adjacent agents. What this new approach proposes is a distributed ShenoyShafer propagation and distributed lazy

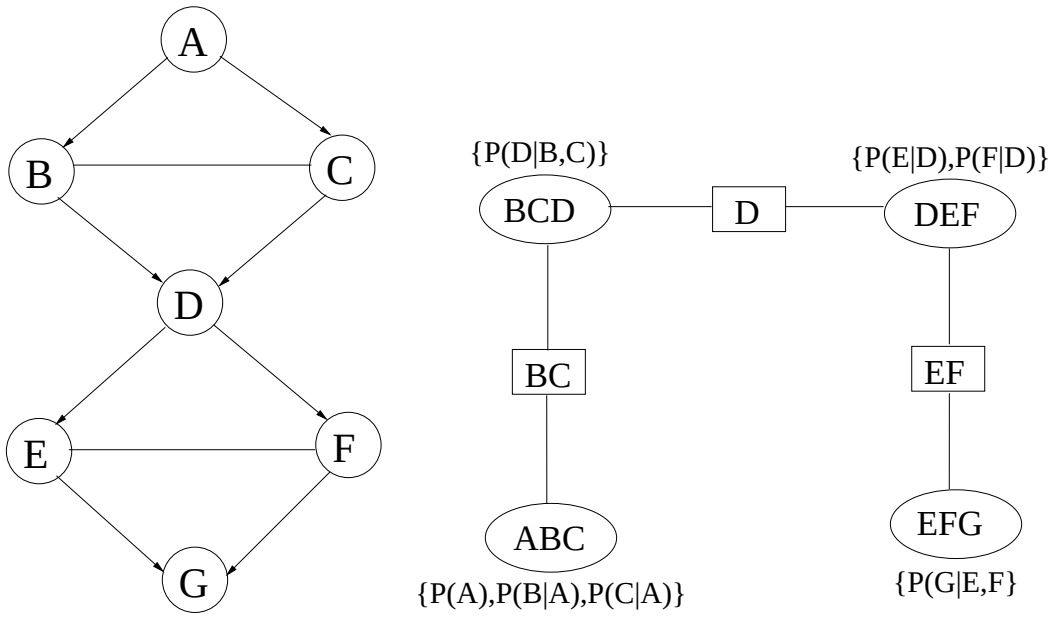


Figure 5.1: Example for Lazy propagation.

propagation to compile the subnet at an agent A_i into a set of JTs $T_{j\to i}$, one for each adjacent agent A_j. The JT will then be used for message passing with the agent.

Let us go back to the step of Distributive Triangulation in the MSBNs inference. In the collect phase, when an Agent A_i calls method Depth_First_Eliminate, when the current agent does is basically the following:

1. Triangulates its graph (G_i) using an order $\sigma_{ij} = (V_i \setminus V_j, V_j)$ for every A_j adjacent to it and different from the caller, adding the fill-ins and sending them to A_j.

2. When the recursive calls to all A_j return, A_i adds to its graph (G_i) the fill-ins received from its adjacent agents.

3. With an elimination order $\sigma_{ic} = (V_i \setminus V_c, V_c)$, triangulates this graph, adding the corresponding fill-ins[2]

4. Sends to the caller agent all fill-ins over the interface nodes between these two agents, I_{ic}.

Finally the call from the root agent to Distribute_Fill-ins method makes all the obtained fill-ins distributed through the agents' tree.

[2]The elimination sequence σ_{ij} denotes the same as $\sigma_{i\to j}$.

In this new approach the authors use the *classical* tandem of collect and distribute information, in this case, fill-ins. So, the process of triangulation is controlled in a different way:

1. From the root (A_{root}) go deeper into its neighbours (adjacent agents) until a leaf agent is reached.

2. Now this leaf agent is triangulated with restriction of a elimination order of type $\sigma_{leaf,caller} = (V_{leaf} \setminus I_{caller}, I_{caller})$

3. This leaf agent sends to the caller agent all fill-ins over the interface nodes between these two agents, $I_{leaf,caller}$.

4. Once an intermediate agent A_i has received all the messages from its deeper neighbours it can send up the corresponding message to the agent caller $I_{i,caller}$, after having been triangulated with an elimination order $\sigma_{ic} = (V_i \setminus I_c, I_c)$. Here the Collection phase finishes.

5. Again from the root agent, a distribution of fill-ins takes place. A_{root} will send all its neighbours A_i the fill-ins resulting of $\sigma_{root,i} = (V_{root} \setminus I_i, I_i)$

6. The previous step is repeated for every node until a leaf is reached.

7. To finish each non-root hypernode A_i performs one more triangulation process as if they are the root:

   - Add to G_i fill-ins received from each neighbour.
   - Eliminate $V_i \setminus V_j$ and add fill-ins to G_i.
   - Set message to G_j as all fill-ins over I_{ij} obtained above

Figure 5.2 represent an example for this method. Arrows indicate fill-in messages sent.

Then Xiang and Jensen [132] study the impact of choosing a certain agent as root. To experiment this behaviour of the fill-ins received depending on the direction of the fill-ins messages, we are back to a previous triangulating example.

In figure 4.6.(a) there is an example MSBN. If we moralise this we obtain the graph of figure 5.3 which is ready to be triangulated.

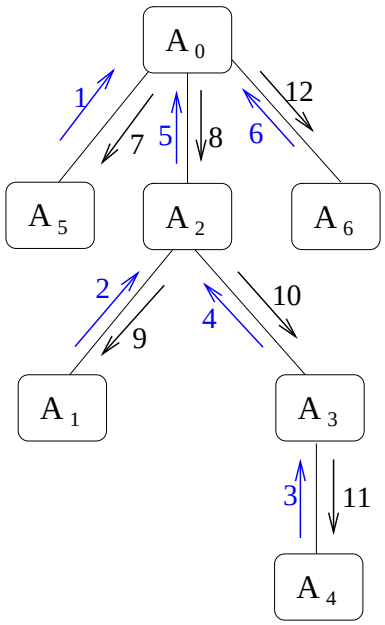


Figure 5.2: An example for Collect & Distribute fill-ins. Arrows in blue (upwards) indicate the collection phase while black (downwards) arrows indicate the distribution stage.

Let us assume that $A1$ is the root for the fill-in propagation. Then fig. 5.4 represent a sequence of steps for triangulating: *downwards* arrows are collection of fill-ins while *upwards* indicate distribution.

We denote as $G^*_{i\to j}$ as the triangulation that takes place when agent A_i sends fill-ins to agent A_j. G^*_i indicates the local triangulation for the agent A_i.

Thanks to this full propagation method we avoid performing n different triangulations for every graph, being n the number of agents in the hypertree. It can be shown that the fill-ins for the inter-agent messages are not dependent of the particular elimination orders used in the triangulations, but on the chosen root for a collection phase, and that is why the previous method was designed like that.

The most important conclusion from this fill-in propagation scheme is how the direction of the message changes the result. So, sometimes G^*_i will be sparser than, for example, $G_{i\to j}$, as happened with G^*_3, and usually this happens the other way around. This characteristic is what the authors wanted to exploit.

To do so, they have slightly modified the way that compilation is carried out for MSBNs. As we have differentiated between G^*_i and $G_{i\to j}$, we could too differentiate a Join Tree T^*_i and another one(s) $T_{i\to j}$. The purpose is to take advantage of the sparse

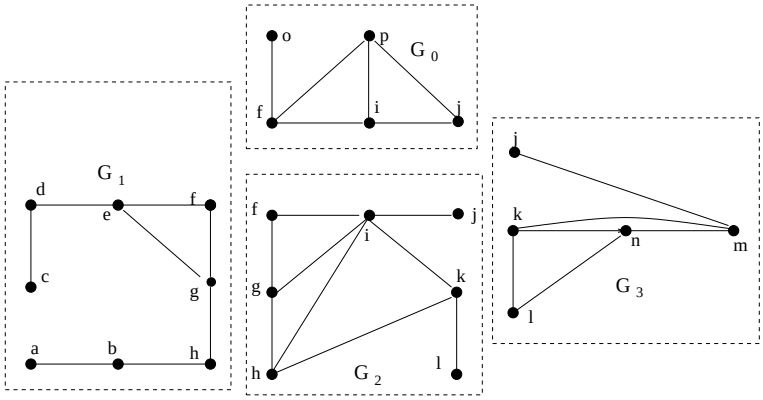


Figure 5.3: Moralised MSBN to perform propagation of fill-ins.

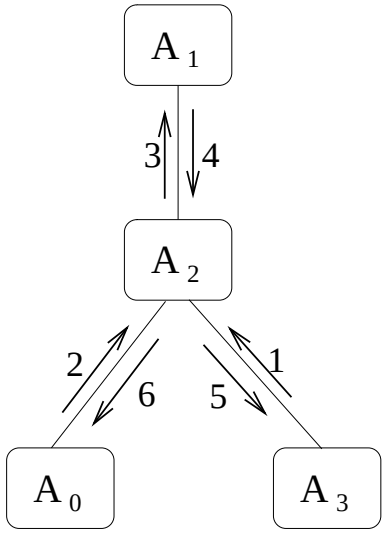


Figure 5.4: Sequence of steps for fill-in propagation in the example if $A1$ is the root.

graphs. We already know that a less dense graph will yield a less complex tree and that will redound to the benefit in a reduction of the computation time.

After triangulation, in this new approach compilation will convert each G_i^* into a Join Tree for local inference and every $G_{i\to j}^*$ into a junction forest (JF) for computing messages from an agent A_i (subnet S_i) to the agent A_j in the inter-agent belief propagation. Then, JFs will be message passing structures whereas JTs will be local inference ones.

Figure 5.6 shows some JFs and a JT for the previous example. Notice that Junction Forest $T_{1\to 2}$ can be basically broken apart by means of separator g joining $[g, h]$ and $[e, f, g]$ because in graph $G_{1\to 2}*$ since the d-sepset in this case $\{f, g, h\}$ is not complete

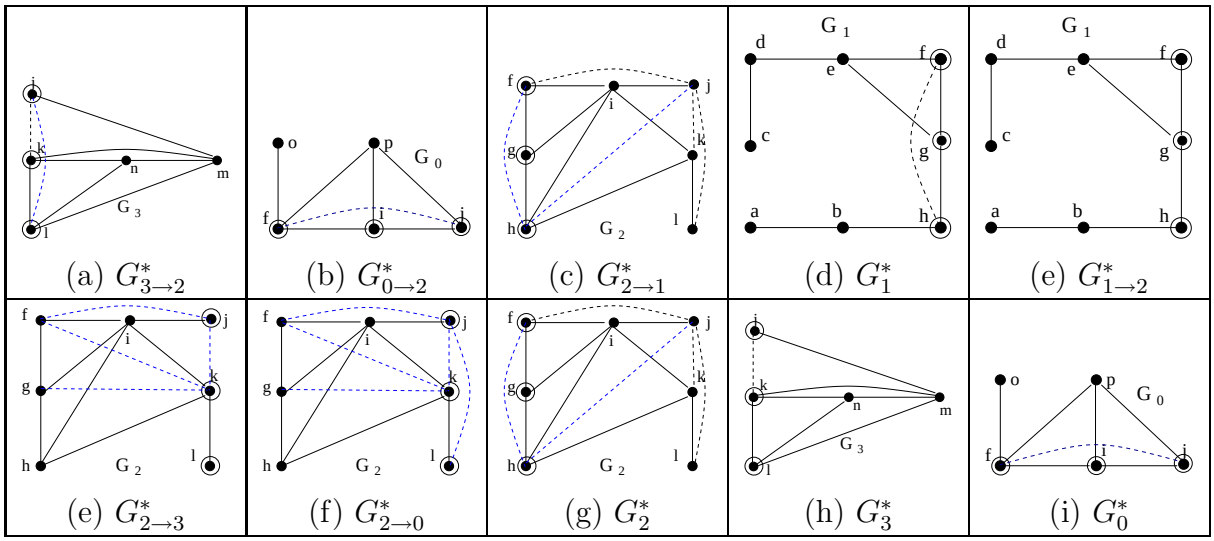


Figure 5.5: Example for the Xiang & Jensen method of propagating fill-ins

and then messages are decomposable. For submessages during inference, those over $\{g, h\}$ can be computed using the equivalent clique ($[g, h]$) and for those over $\{f, g\}$ we will use clique $[f, g, h]$ than contains this set of variables.

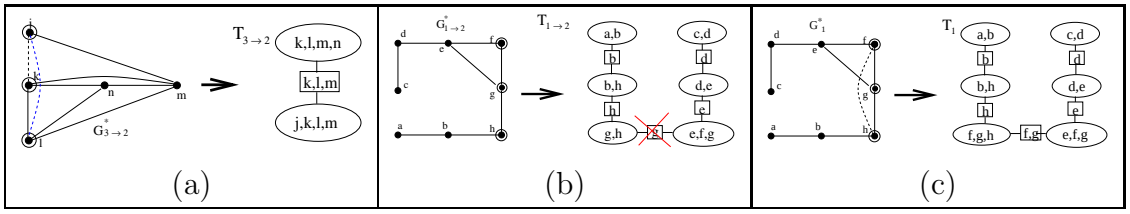


Figure 5.6: Formation of JFs $T_{3\to 2}$ and $T_{1\to 2}$ together with JT T_1.

Then, for these inference messages again probability propagation (Collect & Distribute scheme) will be applied, in such a way that:

- Messages from A_i to A_j (adjacent agents in the hypertree) will be computed by means of $T_{i\to j}$.

- Once an agent A_j receives this message it will be processed by T_j and every $T_{j\to k}$ (being $A_k \in adj(A_j)$ and $k \neq i$). In this sending/reception of messages, again the element of Linkage appears. For example, two linkages are shown in figure

5.7 from $T_{3\to 2}$ till T_2 (complete d-sepset) and from $T_{1\to 2}$ till $T_{2\to 0}$ (incomplete d-sepset).

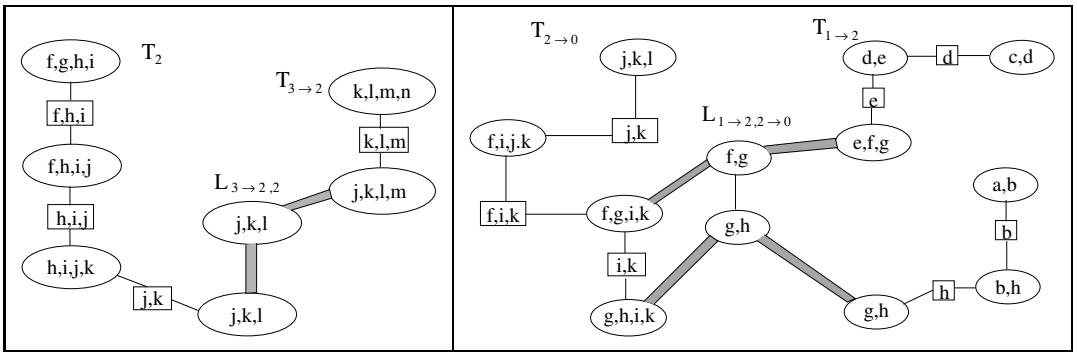


Figure 5.7: Two examples of linkage tree with this alternative method.

About assignment of conditional probability tables, they will be done in a similar way to that in *regular* inference of MSBNs. It could happen that a subnet is associated with various JFs, but only one copy of each CPT will need to be stored.

Then, the main benefit of this variation is that inference can be quite more efficient, since at each step (belief messages) we get as simpler structures as possible, and more simplified messages.

Besides, as mentioned at the beginning, an application of the previous method is the possible adaptation to use other inference architectures as Shenoy-Shafer or Lazy Propagation:

1. For *Shenoy-Shafer*.- The main procedure will do a collection SS propagation for every $T_{i\to j}$ at the host linkage A_j, and then send a message as the marginal of host belief to the linkage. After this extended SS propagation, also SS propagation will be needed in the JTs for local inference.

2. For *Lazy Propagation*.- This will be implemented as an extension of the previous one, but avoiding multiplication of CPTs in an explicit way. For JFs SS propagations will be replace by Lazy Propagation method. Thus, messages over sep-sets and those over linkage will be constituted by a set of belief tables over the corresponding set of variables without being multiplied. The main advantage of this technique, as in the BNs case, the reduction of space complexity given by the decomposition of messages and tables.

5.4 Plug & Play OOBNs

Most of the topics reviewed and studied in the previous sections motivated the work in [6]. Those similarities between MSBNs and OOBNs, the initial lack of efficient inference methods for OOBNs, the coherent theory behind MSBNs definition and their well studied inference methods (including the powerful variation explained in subsection 5.3) and, finally, the design of our strategy of Incremental Compilation to fast re-triangulations were (among others) elements that inspired us [6] to undertake the challenge of creating a way of compiling OOBNs in a more dynamic and efficient manner.

The inference method we have developed will use as tool the propagation of fill-ins, inspired (yet different[3]) by Xiang and Jensen's algorithm described previously. The main element that enables us to do this propagation in a efficient way is the Instance Tree (IT), which always exists and can also be easily and directly obtained from the OOBN (see alg. 22). Besides this propagation scheme, other important improvements for local triangulations have been introduced that will both enhance efficiency and maintain the reusability principle inherent in an OO framework.

What our technique does is a redefinition of the compilation process of an OOBN which has been proved to be valid, and then a correct alternative for inference. But even more important is that this new proposal for OOBNs compilation has yielded two main interesting contributions: (1)This inference can be done directly over an OO structure, since the IT maintains (and differentiates) the various abstraction levels of the corresponding network. (2)It also incorporates a mechanism that allows dynamic/changing OOBNs to be compiled in a very fast way. That is why we chose the name *Plug & Play*, since it allows addition/deletion/modification of classes in a dynamic and quick way. Also, as we will discussed later, this scheme will be an advantage if we decide to *pre-compile* OOBNs when they are being specified, exploiting the fact that we could have several instantiations of a certain class.

To describe our technique, we will divide it into two different problem layers:

- The global problem, which is delimited by the corresponding OOBN, that is, the encapsulating class. Here the Instance Tree will monitor which and how fill-ins are propagating through the tree. For this fill-in propagation the internal instantiations will have to be triangulated with a constrained elimination order.

[3]Since the nature of MSBNs and OOBNs are different, and also their secondary structures (Hypertree and Instance Tree respectively)

- The local compilations, that will give rise to local inference structures obtained from (locally) good elimination sequences. Moreover, they will be benefitted from using Incremental Compilation to speed up re-triangulations.

The next two points expound better how these two layers are tackled in this particular approach.

5.4.1 Global compilation of the OOBN by means of IT-based fill-in propagation.

As seen throughout this work, the compilation process of a BN is the process of transforming it into JT (secondary structure for inference). The purpose of this method is to find a way of obtaining this Junction Tree without renouncing the potential of the modularity capability of OOBNs. As a result, our first concern is the construction of a join tree that keeps the set of instances in the OOBN partitioned as originally. To achieve this goal two conditions must be guaranteed:

- There will be no moral links between variables not in the same instance of the Instance Tree IT[4].
- During triangulation there will be no fill-ins between variables not in the same instance of the IT.

First requirement is always true, since directed links are never between two variables not in the same class[5]. Links between instances are restricted to interface nodes, and these interface nodes will be together in one of the connected instances. Then, when a moral link appear it will also be between variables in this same instance.

Regarding second point, this will be assured by the procedure that we have come up.

IT-based fill-in propagation

To see details about our scheme for propagation of fill-ins algorithms 24 and 25 are written below. Basically it consists in doing a collect phase of fill-ins from the root of the Instance Tree. This collection will be done, as usual, from leaves to the root, where

[4]Regular links are not possible by the proper definition of OOBN.

[5]Note that one variable could be in two different instances only if it belongs to their interface

the elimination sequences are restricted to remove last those nodes in the interface to the next upper instance in the IT (caller instance). Once these two algorithms are executed, we will have a correct triangulation of our OOBN.

Algorithm 24 IT-based Triangulation over an OOBN.

```
1: procedure IT-BASED_TRIANGULATION(OOBN o_net)
2:     Instance_Tree IT ← o_net.get_Instance_Tree()
3:     Instance_root ← IT.get_root()
4:     Instance_root.COLLECT_FILL-INS(null)
   ▷ From the instance root of the IT perform a collect phase of fill-ins.
5: end procedure
```

Algorithm 25 IT-based Collection of fill-ins.

```
1: procedure Instance_now.COLLECT_FILL-INS(Instance_previous)
2:     Fill-ins ← ∅
3:     for all Instance I_j ∈ IT_Down_Adjacent(I_n) do
4:         Fill-ins ← I_j.COLLECT_FILL-INS(I_n) ∪ Fill-ins
           ▷ Fill-ins are collected (propagated) from the downwards adjacent neighbours in the IT.
5:     end for
6:     σ_np ← (I_n.Nodes \ Interface_np, Interface_np)
7:     links ← I_n.Moral_Graph.triangulate(σ_np)
8:     return links^↓Interface_np
9: end procedure
```

Since the interface sets between two instances in the IT d-separate them, we can apply fill-in propagation as explained in [132]. Note that we will only propagate fill-ins, moral links, and regular links between interface nodes (for ease of exposition we will call them all fill-ins). We fix the root of the tree to be the the root instance of the IT, and only a collect of fill-ins will be performed.

Any elimination order for the leaves of the IT can be used, with the constraint that the input nodes and output nodes are not eliminated. Looking at the schematic figure 5.8, the nodes $\{x, y\}$ and z are not eliminated from the instance C. It is well known that the fill-ins between nodes not yet eliminated will be the same no matter the sequence so far, so any order will suffice in this step. This sequence yields a set of fill-ins and those involving two interface nodes are propagated upwards in the IT. Note that some input nodes may not be in the interface, these will be eliminated last, but

before fill-in propagation. In the example, the fill-in between x and y is propagated to B. This process is repeated recursively until the root node has received fill-ins from all its descendants. Using this kind of fill-in propagation we obtain a valid triangulation [132]. This method of triangulation ensures that there are no fill-ins between variables not in the same instance as any fill-in propagated will be between interface nodes contained in the sending as well as the receiving instance in the IT.

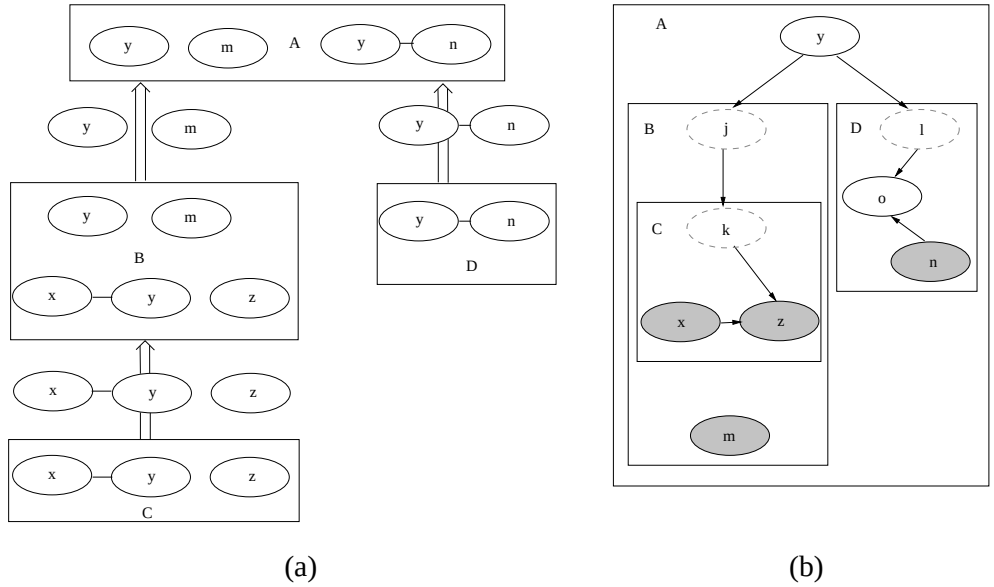


Figure 5.8: (a) An IT where the fill-in messages can be seen. Note that y is in both A's interface to B and to D. (b) An OOBN inducing the IT. Note that the input nodes are not in the IT, as they are overwritten by y.

From this triangulation it will be possible to construct a valid join tree for the global structure as explained in [7]. So, the goal of obtaining a valid way to infer, but maintaining the natural partitioning in trees, has been attained.

Once fill-ins have been properly collected, we can triangulate each instance separately as the interfaces d-separate an instance from the instances in the IT above it. The only requirement is that the interface to the parent in the IT is eliminated last. Time may be spent on finding an optimal triangulation of each instance in the IT under this constraint. Optimal triangulation of the instances will, however, not guarantee an optimal global triangulation. Instead we will look for a way of decreasing

the re-triangulation task if changes occur in the OOBN. Incremental compilation for the local triangulations will be used, and it is explained in the next subsection.

5.4.2 Local re-triangulations using Incremental Compilation

The second great utility of this method is how instance triangulations can be improved locally and, what is more important, how they will permit quick recompilations.

The MPSD-based Incremental Compilation (chapter 3) is going to be extended in order to contemplate a couple of new operations: addition/deletion of fill-ins.

In fact, this extension is not hard to do. Adding a fill-in will be similar to add a link between a node A and B, $A \rightarrow B$, but without caring about moralisations. Similarly, removing a fill-in is like removing a directed link, but again without tracing which moral links should also be removed. Then, they could be seen as simplifications of already existing IC modifications.

As mentioned before, our method not only infers properly without losing the OO structure, but fast local re-triangulations make possible to manage efficiently those networks in dynamic/changing environments. All the changes affecting classes inside the OOBN will be fast and easily processed thanks to the Plug & Play scheme, that we describe next.

5.4.3 Plug & Play behaviour

There will be certain environments where OOBNs suits perfectly such as automatic model generation, dynamic domains, time-slice structures, complex problems,... To manage this kind of particular situations efficiency problems must be solved. In automatic model generation, new data will be received, and then addition/modification of classes has to be done in a quick way. Dynamic domains will be continuously registering new situations. In temporary structures, different slices are incorporated along the time passes. And obviously, for complex domains, a total recompilation of the OOBN will generally be unmanageable.

For all the previous, we found necessary to provide a way of making the corresponding changes to an OOBN by means of a powerful and really dynamic scheme. After having used the two-step method explained before: Propagation of fill-ins through the Instance Tree + Local Triangulations and thanks to the fast local re-triangulations provided by IC we have designed a general Plug & Play technique over OOBNs that manages rapidly with any possible change that could be made in the network.

Only nine (four) different kind of changes are possible:

1. Adding/removing a regular link/node.

2. Adding/removing a reference link.

3. Adding/removing an instance.

4. Changing a class specification.

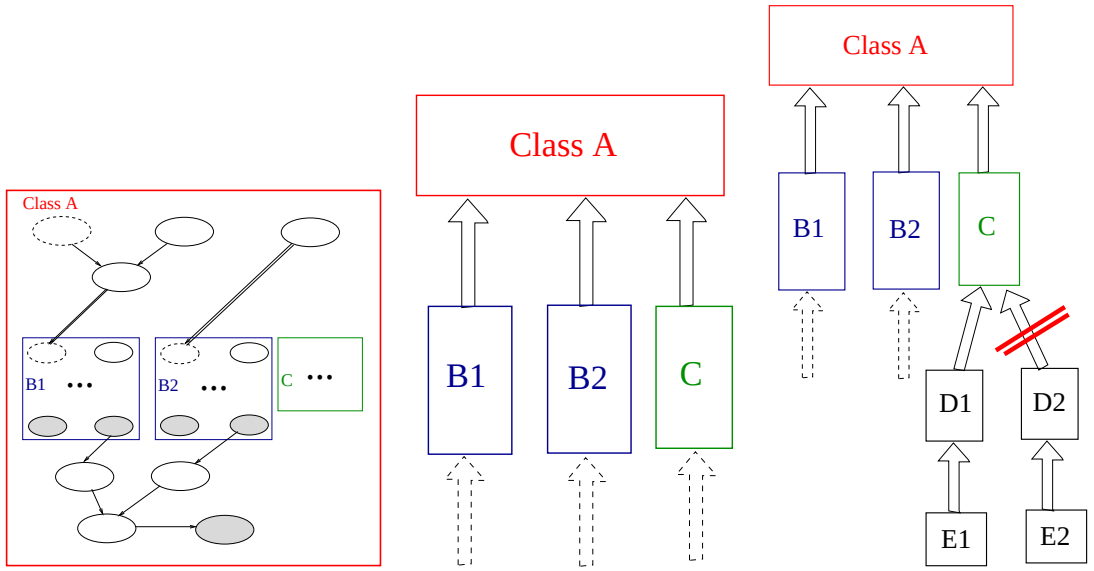
The first eight (three) can only occur in the outermost class, otherwise it will fall under the last. This is because a class only *sees* its protected nodes and the interface to other instance, which consequently are also inside the outer class.

Note that fill-in propagation only occurs in the fourth case, as the others happen in the root instance of the IT, and fill-ins are only propagated upwards in the IT when a class is redefined (re-specified).

Adding/removing a link/node will force a re-triangulation of the root instance, but using IC, only the relevant subset of the graph will be re-triangulated. This will not change the triangulation of instances in the root instance, as these can be triangulated independently of the rest of the network due to the fill-in propagation scheme. If moralisations between output nodes of an instance appear/disappear, the moralisations will only be relevant for the root instance. Figures 5.9.(a)-(b) explain it graphically on a concrete example.

Adding/removing a reference link will not affect the triangulation of the instance where the child of the link is defined. The triangulation of all instances of a class are identical, except for names of reference nodes. Notice that the class of an instance is compiled independently of the uses of the instances. If a reference link is added, the *child* (referenced node) of the reference link will have its name changed, and the default potential for that node, which was used previously, will be removed from the JT. If a reference link is removed, the name of the child will also change, and the default potential will be added to the JT. None of these modifications changes the triangulation.

However the interface to the parent in the IT will change, and hence the message sent upwards during fill-in propagation may change. Adding a reference link will add the parent of the link to the interface to the instance of the child. Removing a reference link will remove the parent from that interface. Changing the interface like this will add (or remove) the fill-ins already found by the propagate fill-ins step for the instance.



(a) (b) (c)

Figure 5.9: (a)A schematic example of OOBN. (b)Its corresponding IT. If we make any modifications on the definition of class C, for example, they will be performed there. Notice that class A does not have access to C's protected nodes. So, the only concern is to test if the communication of links in the collection phase, over the interface, has changed and if so, we should update that information.(c)Suppose that instance $D2$ is removed from C. Then, we just prune the corresponding branch in the IT and ignore the previously collected links from this branch (removing an instance).

If the message is changed, a re-triangulation of the root instance using IC will be triggered. Again, only the relevant part of the root instance will be re-triangulated.

In Fig. 5.10.a an example of an OOBN is shown on top of the triangulation of the IT and the fill-ins propagated. The JT for this OOBN will have the potentials for A, B, Y and Z. If a reference link from B to Y is added the name of Y will change to B in the triangulation, it will be added to the interface to the root instance of the IT, the fill-in will be propagated, and the potential for Y will be removed from the JT as Y no longer exists in the IT. This situation is shown in Fig. 5.10.b. If the reference from B to Y is removed, the situation will return to that in fig. 5.10.a.

Adding/removing an instance will not affect the local triangulation of the instance, as all classes will be triangulated when their specification is done. So the

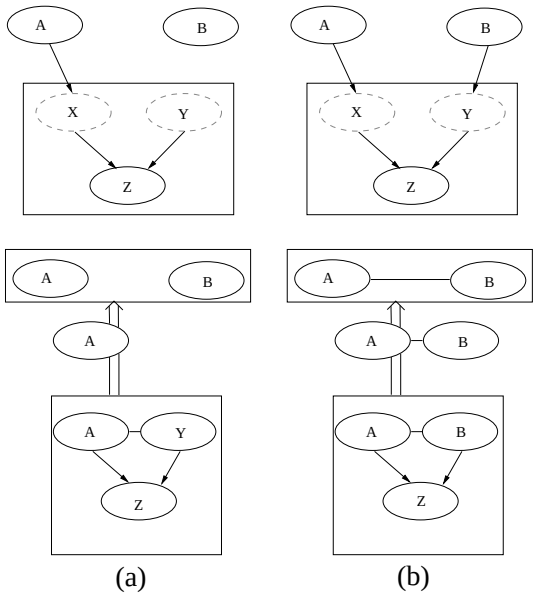


Figure 5.10: (a) An OOBN and the triangulated IT achieved from it. (b) A reference has been added and the triangulation of the IT has changed.

fill-ins induced by them will be known before they are instantiated. The only parts not known are the reference links to input nodes, and some of the children of the output nodes (those children outside the class specification). Note that the triangulation of the instance will not change even if new moralisations occur, as these only affects the encapsulating class. The names of input nodes may change if reference links are added, but this is handled at that time.

Adding an instance with no links will not induce any fill-ins in other instances in the IT, as the instance will be separated from the rest of the network until regular links and reference links are added. So adding an instance will not require any fill-in propagation or re-triangulation as this will be handled when links are added to or from the instance (that would go in the previously described kind of operation). The IT will be updated with the new sub-tree, and the overall JT will have the JT for the instance added, and in effect become a forest.

Removing an instance will be handled by removing the sub-tree for the instance from the IT, and removing the JT associated with it from the overall JT. The root instance will be re-triangulated without the fill-ins received from the removed instance using IC (see fig. 5.9.(c)).

Changing a class specification can include any of the previously mentioned actions any number of times, so to avoid fill-in propagation more than once, propagation

will not occur before modification of the class is completed. The changes to the class are handled by viewing the class as the root instance of its own local IT and using the techniques described above.

Once modification of the class is finished, the fill-ins between the interface nodes are found. The class will already be triangulated and the JT found by this triangulation will substitute the JT part associated with each instance of the class in the overall JT. Note that some names may be changed in the various instances, through reference links, this should be handled when substituting the JT part.

All that remains now is to propagate the fill-ins. An instance only needs to send a message if its fill-ins have changed. The propagation starts in the leaves of the IT and stops when no instance can send a message. This is done by sending a message from the instance in the IT that:

1. Has different fill-ins.
2. Has not sent the message yet.
3. Is farthest away from the root.

Every time an instance receives a message, the fill-ins to its parent will be found and compared to the previous fill-ins to see if a message should be sent further upwards, and it will be re-triangulated using IC. Once no more messages can be sent, the OOBN is triangulated again, using only local operations.

In this scheme it is worth noting that all the information needed to re-triangulate any instance will be available locally in each instance after propagation of fill-ins has been done. Classes can be precompiled, and the JT obtained through this can easily be incorporated into a JT for an OOBN that adds an instance of the class.

5.5 Other possible benefits from the modular nature of IC together with the previous ones.

In chapter 3 we showed the method of MPSD-based Incremental Compilation [43, 44]. In this fourth chapter we have explained other *modular* structures that shared with IC the principle of dividing a large problem into several sets of smaller ones in order to get better performance and to improve efficiency. Of course, if we examine the two main frameworks revised here (MBSNs and OOBNs) together with IC, we observe how these divisions are of different origin and utility, and yet alike in some aspects:

1. MSBNs approach deals with a multi-agent paradigm where, by definition, agents are independent elements that should collaborate, but still they do not show each others, for reasons tied to the own problem's characteristics. For example, as in the logical circuit, to hide certain valuable or private data. But it could be also caused by the impossibility or the cost of communicating the whole information of every agent. In general, as said before, they are thought to solve multi-agent schemes.

2. OOBNs were initially designed to manage large Bayesian networks, with the aim of modelling them in a much easier way that could provide the net builder with the Object Oriented approach, a quite successful methodology for so many other technical fields. They were proved to also enhance other particular needs as dynamic domains, temporal networks or repetitive structures. We have also seen how some MSBNs' procedures have inspired researchers to design specific techniques that improve considerably the process of inference in OOBNs, which makes them even more powerful.

3. Incremental Compilation makes the same process of dividing the problem, but this time in the opposite way. From the big structure, we extract its components, while the previous two constructed incrementally a whole structure *assembling* a set of basic units. IC's point of departure is a unique Bayesian network, normally of great size and complexity, that is quite hard (in time and computationally) to be processed as a complete unit. Besides, this unitary treatment is not even necessary most of the times, since the processes over the net would normally be focused on a certain and localised part of the network.

   Then, IC uses a well-known decomposition method that was proved to maintain interesting properties, such as allowing independent triangulations for subgraphs. And, as already explained, this division has leaded us to design not only a new technique for triangulation (chapter 2) but also a completely new approach for compiling Bayesian networks while editing them that saves time and computation considerably.

The difference among the three listed approaches lies in the way we get the partitioning of the problem, but in the three cases we tackle a complex problem (multi-agent, large domains, big networks) splitting it into smaller pieces that we can work easily and quicker with, having also a manner to re-integrate them when necessary. What this

particular section attempts to demonstrate is that taking as a basis their resemblances, but having in mind their diversity these three *modular* structures can take advantage one from the others to solve some of their own weaker points or just to improve determined methods. It has already been mentioned the link MSBN-OOBN. We want now to indicate more particularly those *benefits* related to our own development, IC.

The first *benefit* is clear and has already been commented here: the integration of IC in a modular and dynamic way of growing (or branching)the OOBNs: the so-called *Plug & Play* OOBNs. It also reinforces the main purpose IC is thought for, which are large and complicated domains.

On the other hand, the next step could be seeking the analogies and the bonds between MSBN and IC. We really think there are and that they could both benefit from the other. Next, we will expose the reasons for that statement, supported by an analysis of similarities and differences.

Firstly, as we have shown IC is a method for making more efficient the (re)compilation of a Bayesian network. We also have revised how MSBNs are an extension of BNs, and because of that their compilation process, which we called Cooperative Compilation, was inspired in the single agent paradigm. In fact, there was a clear parallelism between both of them. So, the following ideas comes naturally: (1)could MSBNs use the decomposition in SPMs? (2)If so, could this decomposition be of utility for triangulation methods or for performing an *adapted* IC method?

If we think about first question, we could also look for the link MSBN-IC going into their basic elements, which is the relation between an agent and a prime subgraph. If an interface I_{ij} between two agents A_i and A_j is completely connected (which could easily happen as some examples illustrated), then in fact these two agents together with the I_{ij} set will be correspond to three maximal prime subgraphs. That makes possible to triangulate them independently, without needing any fill-in propagation. Even if this interface I_{ij} is not complete, another way of constructing SPMs will be aggregating several agents in an only subgraph.

From the previously explained, it is possible an application of the parallel/independent triangulation scheme shown in chapter 2 in order to speed up the triangulation process. We should remind that this process could became specially complicated in an MSBN environment. Thus, any optimisation for it will be of great interest. Obviously, the subgraph-level triangulation will differ from the *regular* one when a subgraph covers several agents. In this case, it would be necessary to implement a variant algorithm of the *Cooperative Triangulation*. This adaption can be seen as a partial Cooperative

Triangulation and, at the same time, we would have to adapt the IC method to the new scenery. We are not dealing exclusively with join trees any more, but also with linked junction forests and their intermediate structures for communication: the linkage trees. So, this Multiply Sectioned IC will need a mechanism able to manage this new trees, still based on the original JT, which gives us the confidence that a design of this transformation scheme is quite possible. So, this first cooperation MSBN-IC could be seen as an application of the Incremental Compilation to the original compilation scheme for MSBNs, which seems attractive due to the complexity of the multi-agent domains and problems.

Moreover, from a different point, there is a second advantage that the tandem IC-MSBN can imply. In this case the Incremental Compilation procedure could be itself benefitted. One of the main drawbacks for the MPSD-based IC is that this is quite dependent on the obtained decomposition. It is clear that subgraphs are the smaller component we can treat. Hence, when a very large subgraph comes from the decomposition, we still have to tackle it as a complete unit. In some cases, it could be really a problem, because the MPSD can return an unbalanced decomposition, which makes our *modular* tool less powerful. That is why we launched in previous chapters the idea of using in those cases a kind of approximate IC. That implied breaking big subgraphs in smaller pieces, but also dealing with the fact that the MPSs properties were not applicable in those sub-divisions. We are investigating the possibility of using α-MPSs, where α is a ratio of completeness. That is, if we have a separator containing 4 nodes, it will divide the graph into two MPSs only if the projected graph is completely connected, which means that every node is linked to the rest (6 links in total). We could consider *quasi*-complete subgraphs, i.e. in the previous example if 5 out of the 6 links exist, the corresponding subgraph will be said to be 0.83-complete ($5/6 = 0.83$). And then it is also 0.80-complete, for example (see def. 31)

Definition 31 A graph G, being $G = (V, E)$, is said to be **α-complete** if

$$\frac{|E|}{|E^c|} \geq \alpha$$

where $\alpha \in (0, 1]$ and $|E^c|$ represents the number of links for $E^c = \cup_{i \neq j}\{V_i - V_j\} = \frac{|V| \cdot (|V|-1)}{2}$, that is, there is a link between every V_i and V_j, being $G^c = (V, E^c)$ the corresponding complete graph.

□

Nevertheless, the alternative scheme for this approximation has not been defined yet. In this sense the fill-ins propagation scheme used for MSBNs (and somehow

migrated to OOBNs) could again be useful for the IC technique. Those big subgraphs split into several α-subgraphs can then be triangulated through a propagation scheme. We could basically used the method by Xiang & Jensen over certain portions of the Bayesian networks when they are involved in re-compilations. The main idea is to treat big subgraphs by means of the Cooperative Triangulation method, by an *artificial* MSBN, where we can choose the features of the subnets, their sizes, their common links, the (almost complete) interfaces, and so on. This α-decomposition could also be applied to the *triangulation by re-triangulation* method. If this scheme is used, those subgraphs sub-divided in smaller pieces (subnets) will communicate through this α-MPST as agents did in the hypertree. Of course, as remarked above, it is only necessary to those big subgraphs that present a really unbalanced size and that produce big clusters in the tree. It would also be useful a study about which values of α and under which specific situations (partitions) this technique can present a real advantage.

Chapter 6

Incremental algorithm for performing parcial abductive inference

A little inaccuracy sometimes saves tons of explanation.
H.H. Munro. (1870–1916)
Burman born English novelist and short-story writer, under pseudonym Saki.

6.1 Abductive inference in Bayesian networks. Total and partial abduction techniques

Until here, we have been dealing with inference in Bayesian networks from the most typical and classical approach, that is, looking for the posterior probability distributions of the variables when some evidence has been observed. This is indeed the most common probabilistic inference in Bayesian networks (BNs) where normally probability or evidence propagation [99, 21, 62] is used. In this case, as reviewed, the look for the computation of posterior probability for all non-observed variables given a set of observations ($X_O = x_O$) (the *evidence*) was the main goal. But this is not the only existing view we can have from BNs computation, and there are also other interesting inference tasks.

In this category we can find abductive reasoning, which represents another type of propagation and has also a great relevance within this BNs field. Abduction is a method for inference used for finding/generating explanations to some observed facts. Generating explanations in Bayesian networks can be understood in two (main) differ-

ent ways:

1. *Explaining the reasoning process* (see [68] for a review). That is, trying to justify *how* a conclusion was obtained, *why* new information was asked, etc.

2. *Diagnostic explanations* or *abductive inference* (see [48] for a review). In this case the explanation reduces to *factual* information about the state of the world, and the best explanation for a given evidence is the state of the world (configuration) that is the most probable given the evidence [99].

In this case we will focus on the second approach. Therefore, given a set of observations or evidence ($X_O = x_O$ or x_O in short) known as the *explanandum*, we aim to obtain the best configuration of values for the explanatory variables (the *explanation*) which is consistent with the *explanandum* and which could be assumed to predict it.

As the abductive task is mostly considered as the search of explanations its major application has been done for diagnosis and analysis problems [104, 100], where medical diagnosis stands out especially. In the last years various authors have directed their research endeavours to the study of performing abductive inference for the formalism of Bayesian networks. Two main abductive tasks in BNs are identified[1]:

- *Most Probable Explanation* (MPE) or *total abduction*. In this case all the unobserved variables (X_U) are included in the explanation [99]. The *best* explanation is the assignment $X_U = x_U^*$ which has maximum a posteriori probability given the *explanandum*, i.e.,

$$x_U^* = \arg \max_{x_U \in \Omega_{X_U}} P(x_U|x_O). \tag{6.1}$$

  Searching for the best explanation has the same complexity (NP-hard [119]) as probability propagation, in fact the best MPE can be obtained by using probability propagation algorithms but replacing summation by maximum in the marginalisation operator [29]. However, as it is expected to have several competing hypothesis accounting for the *explanandum*, our goal usually is to get the K best MPEs. Nilsson [91] showed that using algorithm in [29] only the first three MPEs can be correctly identified, and proposed a clever method to identify the remaining $(4, \ldots, K)$ explanations.

[1] We introduced them in chapter 1, but we have considered that it is worth re-writing basic concepts as the same time as we complement them with further detail.

This kind of problems has been studied by several authors who developed exact algorithms [78, 109] and there are also works using approximate methods [50] in order to solve them. Nilsson proved that under certain assumptions, his algorithm to obtain K-MPEs has a similar complexity to the computation of posterior probabilities by Hugin algorithm [91, 90]

One of the main drawbacks of the MPE definition is that as it produces complete assignments, the explanations obtained can exhibit the *overspecification* problem [117] because some non-relevant variables have been used as explanatory.

- *Maximum a Posteriori Assignment* (MAP) or *partial abduction* [87, 117]. The goal of this task is to alleviate the overspecification problem by considering as target variables only a subset of the unobserved variables called the *explanation set* (X_E). Sometimes certain variables clearly have no explanatory value (e.g. in network of figure 6.20 "radio does not work" is not explanatory); others will be just intermediate nodes (e.g. in the network of fig. 6.2 *markTP* is a combination of *Theory* and *Practice*, which are the real, let us say "input" values); and in general there could be variables useless in an explanation because they simply not give any information from an abductive point of view or we are just not interested in them.

  Then, we look for the maximum a posteriori assignment of these variables given the explanandum, i.e.,

$$x_E^* = \arg\max_{x_E} P(x_E|x_O) = \arg\max_{x_E} \sum_{x_R} P(x_E, x_R|x_O), \tag{6.2}$$

  where $X_R = X_U \setminus X_E$.

  In principle it could seem that the solution for this problem would arise by means of an adaptation of those methods developed for total abduction. However, a deeper study of the problem reveals that this does not always hold, and moreover those resolution methods will be in general less efficient. Therefore, algorithms are almost always approximate (such as local search, Simulated Annealing or Genetic Algorithms). This problem is therefore more complex than the MPE problem, because it can be NP-hard even for cases in which MPE is polynomial (e.g., polytrees) [95, 16], although recently Park and Darwiche [94, 95] have proposed exact and approximate algorithms to enlarge the class of *efficiently* solved cases. With respect to looking for the K best explanations, exact and

approximate algorithms which combine Nilsson algorithm [91] with probability trees [108] have been proposed in [17].

In general, we could say that partial abduction has been less studied than total abduction [46].

The problem in both cases is set out as the search of the instantiation values for all (total) or a subset of (partial) non-observed variables in such a way that the joint probability of the configuration is maximum. Besides, since it is not generally enough by finding the best explanation, but we need the K best explanations, this K number is usually added to the problem statement.

The question now is which variables should be included in the explanation set. Many algorithms avoid this problem by assuming that the explanation set is provided as an input, e.g., given by the experts or users. Many others interpret the BN as a causal one and only ancestors of the *explanandum* are allowed to be included in the explanation set (sometimes only root nodes are considered) [78]. However, including all the ancestors in the explanation set does not seem to avoid the overspecification problem and even so, what happens if the network does not have a causal interpretation?, e.g., it has been learnt from a data base or it represents an agent's beliefs [22]. Shimony [117, 118] goes one step further and describes a method which tries to identify the relevant variables (among the *explanandum* ancestors) by using independence and relevance based criteria. However, as pointed out in [22] the explanation set identified by Shimony's method is not as concise as expected, because for each variable in the *explanandum* all the variables in at least one path from it to a root variable are included in the explanation set. Henrion and Druzdzel [55] proposed a model called *scenario-based explanation*. In this model a tree of propositions is assumed, where a path from the root to a leaf represents a scenario, and they look for the scenario with highest probability. In this model, partial explanations are allowed, but they are restricted to come from a set of predefined explanations.

As stated in [22] *conciseness* is a desirable feature in an explanation, that is, the user usually wants to know only the most influential elements of the complete explanation, and does not want to be burdened with unnecessary detail. This follows the logical principle known as *Occam's razor*[2] which can be stated as *one should not increase, beyond what is necessary, the number of entities required to explain anything.* This criterium for deciding among scientific theories or explanations is also said to be

[2]Attributed to the medieval philosopher William of Occam (or Ockham).

parsimonious. Because of this conception of choosing the simplest explanation for a phenomenon, a different approach is taken in [15]. The idea is that even when only the relevant variables to the *explanandum* are included in the explanation set, the explanations can be simplified due to context-specific irrelevance. This idea is even more interesting when we look for the K MPEs, because it allows us to obtain explanations with different number of literals. In [15] the process is divided into two stages: (1) the K MPEs are obtained for a given prespecified explanation set, and (2) then they are simplified by using different independence and relevance based criteria.

In our work we try to obtain simplified explanations directly. The reason is that the second stage in [15] requires to carry out several probabilistic propagations and so its computational cost is high (and notice that this process is carried out after -a complex- MAP computation). Another drawback of the procedure in [15] is that it is possible, that after simplification, the explanations are not mutually exclusive, we can have even the case of two explanations such that one is a subset of the other. Here, our basic idea is to start with a predefined explanation set X_E, and them we build a tree in which variables (from X_E) are added in function of their explanatory power with respect to the *explanandum* but taken into account the current context, that is, the partial assignment represented by the path obtained from the root to the node currently analysed. Variables are selected based on the idea of *stability*, that is, we can suppose that our system is (more or less) stable, and that it becomes unstable when some (*unexpected*) observations are entered into the system. The instability of a variable will be measured by its entropy or by means of its (im)purity (GINI index). Therefore, we first select those variables that reduce most the uncertainty of the non-observed variables of the explanation set, i.e., the variables better determining the value of the explanation variables. Of course, the tree does not have to be symmetric and we can decide to stop the growing of a branch even if not all the variables in X_E have been included. In any case, our set of explanations will be mutually exclusive, and will have the additional property of being exhaustive, i.e., we will construct a full partition of the set of possible configurations or scenarios of the values of the variables in the explanation set.

Along the subsequent sections we will describe our method in detail (section 6.2) and afterwards illustrate it: first (section 6.3), by using some (toy) study cases. These two first points will be a slightly extended version of the published paper [45]. Section 6.4 will do a further analysis on the obtained trees and another example (used in [15]) is added to our experiments in order to compare both techniques.

6.2 On the search for minimal explanations: the *Explanation Tree*

Our method aims to find the best explanation(s) for the observed variables that do not necessarily have a fixed number of literals and we want to achieve that **directly**. The provided explanations will adapt to the current circumstances. Sometimes that a variable X takes a particular value it is an explanation by itself (Occam's razor) and including other variables to this explanation will not add any new significant information.

We have then decided to represent our solutions by a tree, the *Explanation Tree* (ET). In the ET, every inner node will denote a variable of the explanation set and every branch from this variable will indicate the instantiation of this variable to one of its possible states. Each node of the tree will determine an assignment for the variables in the path from the root to it: each variable equal to the value on the edge followed by the path. This assignment will be called the *configuration* of values associated to the node. In the explanation tree, we will store for each leaf the probability of its associated configuration given the evidence. The set of explanations will be the set of configurations associated to the leaves of the explanation tree ordered by their posterior probability given the evidence.

For example, in figure 6.1 we can see three variables X,Y and Z that belong to the explanation set, since they are nodes in the tree. In this particular example there are four leaves nodes, that is, four possible explanations, each of them labelled as Explan_i. What this ET indicates is that, given the observed evidence, that X has the value $x1$ is a valid explanation for such situation (with its probability/weight associated). But if it is not the case then we should look into other factors, in this case Y. For example, we can see that adding $Y = y2$ to the explanation will be enough. Otherwise, when Y takes value $y1$ the node needs to be *expanded* and we have to look for other involved factors in order to find a valid explanation (in this example, variable Z).

Although the underlying idea is simple, how to obtain this tree is not so evident. There are two major points that have to be answered:

- As the ET is created in a top-down way, given a branch of the tree, how to select the next variable?
- Given our goals, i.e. allow asymmetry and get concise explanations, how to decide when to stop branching?

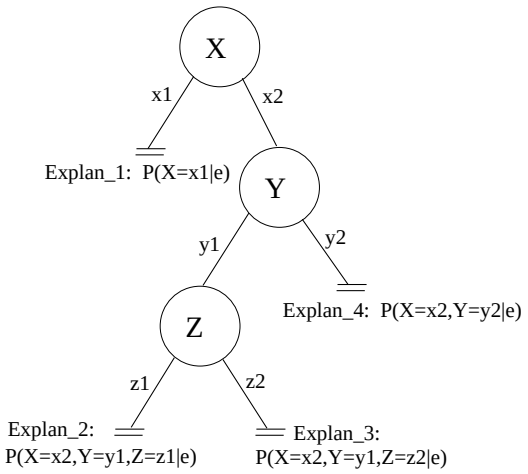


Figure 6.1: An example of explanation tree.

To solve the two previous questions we have used information measures. For the first one, we look for the variable that once instantiated the uncertainty of the rest explanation variables is reduced at maximum. In other words, given the context provided by the current branch, we identify the most explicative as the one that helps to determine the values of the other variables as much as possible.

Algorithm 26 (CREATE-NEW-NODE) recursively creates our ET. In this algorithm we assume the existence of an inference engine that provides us with the probabilities needed during tree growing. We comment on such engine in Section 6.2.1. The algorithm is called with the following parameters:

1. The evidence/observations to be explained x_O.

2. The *path* corresponding to the branch we are growing. In the first call to this algorithm, i.e. when deciding the root node, this parameter will be null.

3. The current explanation set (X_E). That is, the set of explanatory variables already available given the context (path). In the first call X_E is the original explanation set. Notice also that if $X_E = X_U$ in the first call, i.e., all non-observed variables belong to the explanation set, then the method has to select those variables relevant to the explanation without prior information.

4. Two real numbers α and β used as thresholds (on information and probability respectively) to stop growing.

5. The final explanation tree that will be recursively and incrementally constructed as an accumulation of branches (paths). Empty in the initial call.

Algorithm 26 Creates a new node for the explanation tree.

1: **procedure** CREATE_NEW_NODE(x_O,*path*,X_E,α,β,ET)
2: **for all** $X_j, X_k \in X_E$ **do**
3: `Info`$[X_j, X_k]$ $= Inf(X_j, X_k|x_O, path)$
4: **end for**
5: $X_j^* = \arg\max_{X_j \in X_E} \sum_{X_k}$ `Info`$[X_j, X_k]$
6: **if** CONTINUE(`Info`[],X_j^*,α) and $P(path|x_O) > \beta$ **then**
7: **for all** state x_j of X_j^* **do**
8: $new\_path \leftarrow path + X_j^* = x_j$
9: CREATE_NEW_NODE(x_O,new_path,$X_E \setminus X_j^*$,α,β,ET)
10: **end for**
11: **else**
12: $ET \leftarrow ET \cup <path, P(path|x_O)>$ ▷ *update the ET adding* path
13: **end if**
14: **end procedure**

In algorithm 26, for each variable in the explanation set, X_j, we compute the sum of the amount of information that this variable provides about all the current explanation variables conditioned to the current observations $x_O^* = (x_O, path)$. We are interested in the variable that maximises this value. In our study we have considered two classical measures: *mutual information* ($Inf(X_j, X_k|x_O^*) = I(X_j, X_k|x_O^*) = \sum_{x_j,x_k} P(x_j, x_k|x_O^*) \log\left(\frac{P(x_j,x_k|x_O^*)}{P(x_j|x_O^*).P(x_k|x_O^*)}\right)$) and *GINI index* ($Inf(X_j, X_k|x_O^*) = GINI(X_j, X_k|x_O^*) = 1 - \sum_{x_j,x_k} P(x_j, x_k|x_O^*)^2$). Thus, there are different instances of the algorithm depending on the criterion used as *Inf*.

Once we have selected the next variable to be placed in a branch, we have to decide whether or not to expand this node. Again, we will use the measure *Inf*. The procedure CONTINUE is the responsible to take this decision by considering the vector `Info[]`. This procedure considers the list of values `Info`$[X_j^*, X_k]$ for $X_k \neq X_j^*$, then it computes the maximum, minimum, or average of them, depending on the particular criterion we are using. If this value is greater than α it decides to continue. Of course the three criteria give rise to different behaviour, being minimum the most restrictive, maximum the most permissive and having average and intermediate behaviour.

Notice that when only two variables remain in the explanation set, the one selected in line 5 is in fact that having greater entropy ($I(X, X) = H(X)$) if mutual information

(or MI) is used. Also, when only one variable is left, it is of course the selected one, but it is necessary to decide whether or not it should be expanded. For that purpose, we use the same information measure, that is, $\mathrm{I}(X, X)$ or $\mathrm{GINI}(X, X)$, and only expand this variable if it is at least as uncertain (unstable) as the distribution $[1/3, 2/3]$ (Normalising with more than two states[3]). That is, we only add a variable if it has got more uncertainty than a given threshold.

6.2.1 Computation

Our inference engine is (mainly) based on Shenoy Shafer propagation algorithm running over a binary join tree [116]. Furthermore, we have forced the existence of a single cluster (being a leaf) for each variable in X_E, i.e. a clique which contains only a variable. We use these clusters to enter *as evidence* the value to which an explanatory variable is instantiated, as well as to compute its posterior probability.

Here we comment on the computation of the probabilities needed to carry out the construction of the explanation tree. Let us assume that we are considering to expand a new node in the tree which is identified by the configuration (path) $C = c$. Let x_O^* be the configuration obtained by joining the observations $X_O = x_O$ and $C = c$. Then, we need to calculate the following probabilities:

- $P(X_i, X_j | x_O^*)$ for $X_i, X_j \in X_E \setminus C$. To do this we use a two stage procedure:

    1. Run a full propagation over the join tree with x_O^* entered as evidence. In fact, many times only the second stage (i.e., *DistributeEvidence*) of Shenoy-Shafer propagation is needed. This is due to the single cliques included in the join tree, because if only one evidence item (say X) has changed[4] from the last propagation, we locate the clique containing X, modify the evidence entered over it and run *DistributeEvidence* by using it as root.

    2. For each pair (X_i, X_j) whose joint probability is required, locate the two closest cliques (C_i and C_j) containing X_i and X_j. Pick all the potentials in the path between C_i and C_j and obtain the joint probability by using variable elimination [30] (Alg. 5). In this process, we can take as basis the deletion

[3] This normalisation has been done by using $\frac{H(X_i)}{log|\Omega_{X_i}|}$ for entropy (MI) and $\frac{GINI(X_i)}{\frac{|\Omega_{X_i}|-1}{|\Omega_{X_i}|}}$ for GINI index.

[4] Which happens frequently because we build the tree in depth, and (obviously) the create-node algorithm and the probabilistic inference engine are synchronised.

sequence implicit in the joint tree (but without deleting the required variables) and then the complexity is not greater than the complexity of sending a series of messages along the path connecting C_i with C_j for each possible value of X_i. But, the implicit triangulation has been optimised to compute marginal distributions for single variables, and it is possible to improve it to compute the marginal of two variables as in our case. The complexity of this phase is also decreased by using caching/hashing techniques, because some sub-paths can be shared between different pairs, or even a required potential can be directly obtained by marginalisation from one previously cached.

- $P(C = c|x_O) = \frac{P(C=c,x_O)}{P(x_O)}$. This probability can be easily obtained from previously described computations. We just use $P(x_O)$ that is computed in the first propagation (when selecting the variable to be placed in the root of our explanation tree) and $P(x_O^*) = P(C = c, x_O)$ which is computed in the current step (full propagation with x_O^* as evidence).

Though this method requires multiple propagations, all of them are carried out over a join tree obtained without constraining the triangulation sequence, and so it (generally) has a size considerably smaller than the join tree used for partial abductive inference over the same explanation set [95, 16]. Besides, the join tree can be pruned before starting the propagations [16].

6.3 Examples for an initial testing: first study

Because we were in an initial stage of research about the ET method, in order to show how it works and the features of the provided explanations, we found interesting to use some (toy) networks having a familiar meaning for us, to test whether the outputs are reasonable.

We used the following two cases:

1. academe network: it represents the evaluation for a subject in an academic environment, let us say, university, for example. This simple network has got seven variables, as Fig. 6.2 shows. Some of them are intermediate or auxiliary variables. What this network tries to model is the final mark for a student, depending on her practical assignments, her mark in a theoretical exam, on some possible

extra tasks carried out by this student, and on other factors such as behaviour, participation, attendance... We have chosen this particular topic because the explanations are easily understandable from an intuitive point of view.

In this network we consider as evidence that a student has failed the subject, i.e., $x_O \equiv$\{*finalMark*=failed\}, and we look for the best explanations that could lead to this fact. We use \{*Theory, Practice, Extra, OtherFactors*\} as the explanation set. In this first approach we run our ET-based algorithm with $\beta = 0.0$ (i.e. the growing of the tree is not limited when the explanations have very little probability), α=0.05|0.07 and criterion = max|min|avg. Figure 6.4 summarises the obtained results (variables are represented by using their initials).

2. gates network: this second net represents a logical circuit (Fig. 6.3.a). The network (Fig. 6.3.b) is obtained from the circuit by applying the method described in [32]. The network has a node for every input, output, gate and intermediate output. Again, we use an example easy to follow, since the original circuit only has got seven gates (two NOT-gates, two OR-gates and three AND-gates) and the resulting network has 19 nodes.

   In this case, we consider as evidence one possible input for the circuit (`ABCDE = 01010`) plus an erroneous output (given such input), `KL=00`. Notice that the correct output for this case is `KL=01`, and also notice that from the transformation carried out to build the network, even when some gates are wrong the output could be correct (see [32]). So our evidence is `ABCDEKL = 0101000` and we consider $X_E = \{A1, A2, A3, O1, O2, N1, N2\}$ as the explanation set with the purpose of detecting which gate(s) is(are) faulty. Figures 6.5 and 6.6 show the trees obtained for mutual information (I) and GINI respectively. The same parameters as in the previous study case are used but $\beta = 0.05$.

6.3.1 Analysis of the obtained trees

The first thing we can appreciate from the obtained trees is that they are *reasonable*, i.e., the produced explanations are those that could be expected.

Regarding the academe network, when a student is failed, it seems reasonable that the most explicative variable is *theory* because of the probability tables introduced in the network. Thus, in all the cases *Theory* is the root node, and also in all the cases

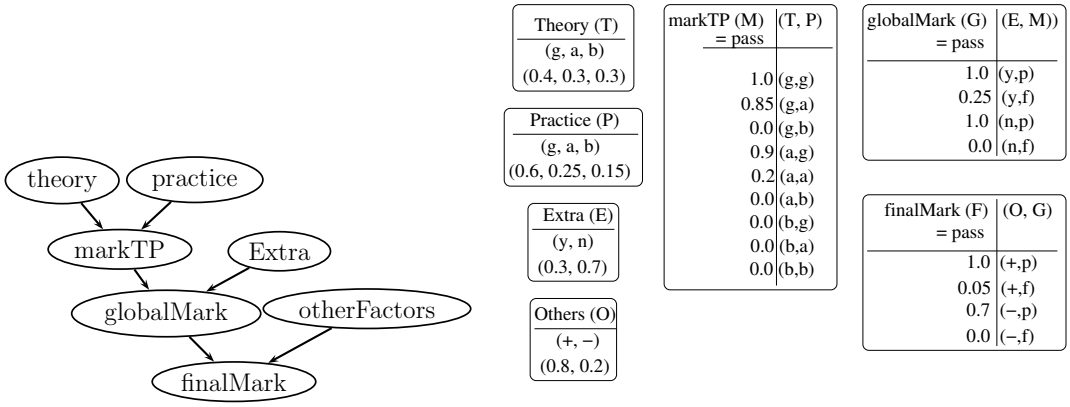
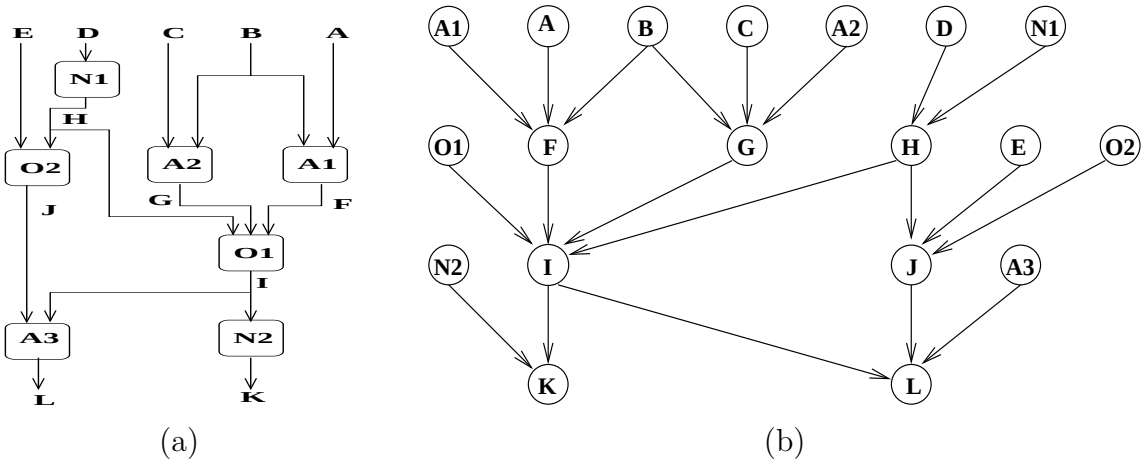


Figure 6.2: Case of study 1: academe network.



(a) (b)

Figure 6.3: (a) Original logic circuit. (b) Network gates obtained from (a) by using the transformation described in [32].

{*theory*=bad} constitutes an explanation by itself, being in fact the most probable explanation (0.56).

The other common point for the obtained ETs is that the branch with *theory* as good is always expanded. It is clear that being *theory* ok another reason must explain the failure. On the other hand, the main difference between the two ETs is that 6.4.(a) expands the branch {*theory*=average} and (b) does not. It is obvious that a bigger α makes the tree more restrictive. If this tree is expanded, as α=0.05 does, is because when *theory* is average it can be interesting to explore what happens with the *practical* part of the subject.

It is possible that variables that are not part of an explanation and that change their 'a priori' usual value or that have an important change in its 'a priori' probability

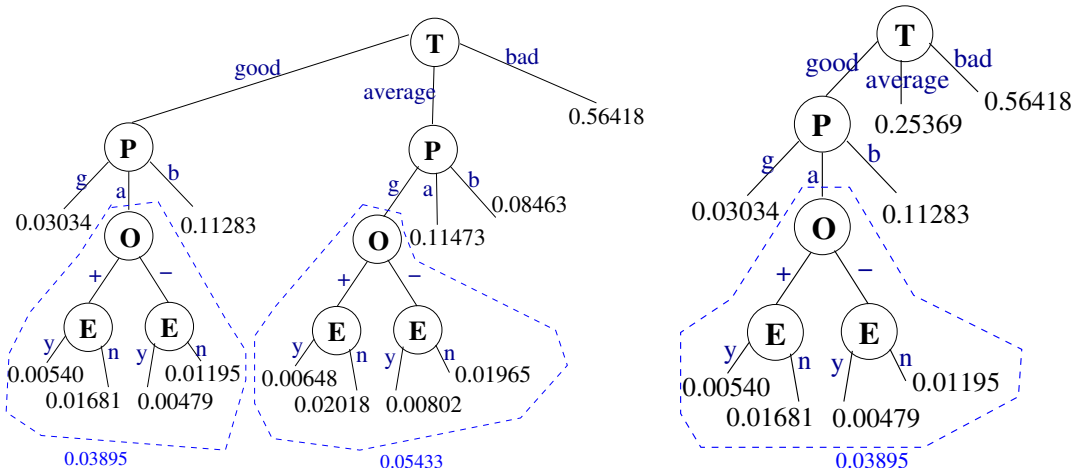


Figure 6.4: Results for academe: (a) is the obtained tree for all MI cases except (MI,α=0.05,min) which produces tree (b) together with all (gini,α=0.05) cases and (gini,α=0.07,max). Finally it is necessary to remark that (gini,α=0.07,min|avg) leads to an empty tree, $\emptyset$, that is no node is expanded. β is 0.0.

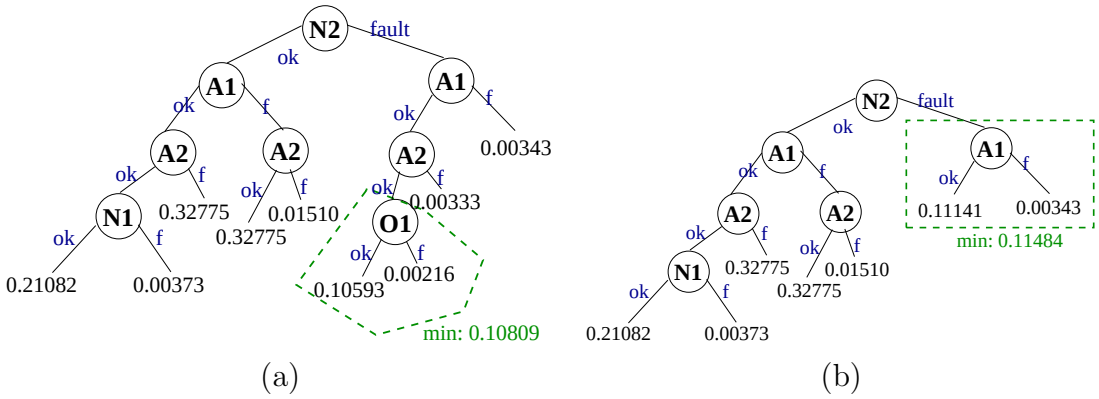


Figure 6.5: Results for gates and MI: (a) is the obtained ET for (MI,α=0.05,max|avg) and also (MI,α=0.07,max); (b) is for (MI,α=0.07,avg). In both cases min prunes more the tree than avg, so the dotted area would not be expanded. β is 0.05.

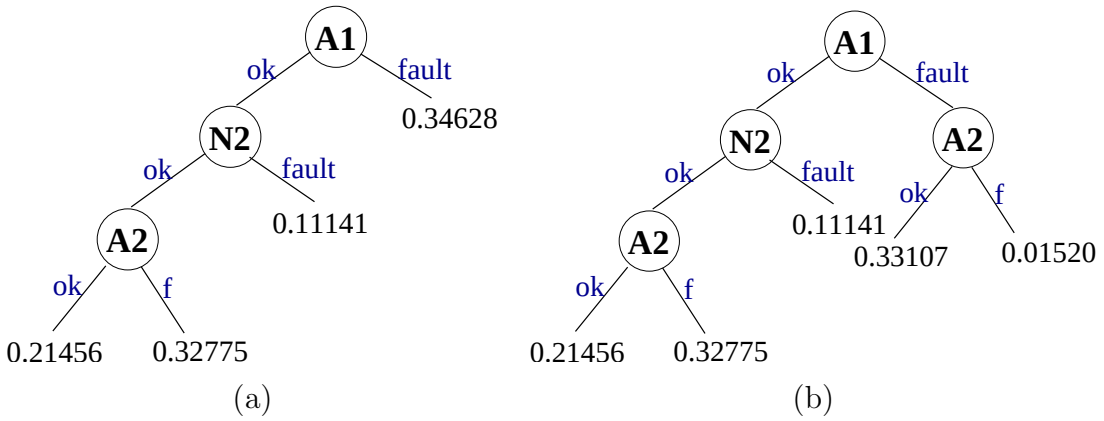


Figure 6.6: Results for gates and GINI: (a) represents the tree for all gini cases, except (gini,α=0.05,max) which produces tree in part (b). β is 0.05.

distribution could be added to the explanation as this could be useful to the final user to fully understand some situations. An example can be the case of academe network with $\{theory = \text{good}, practice = \text{good}\}$. This branch is not expanded. The reason is that in this situation, the other variables have small entropy: *Extra* should be 'no' and *OtherFactors* '-', with high probability. This implies an important change with respect to 'a priori' probabilities for these values, and then these variables with their respective values could be added to the explanation $\{theory = \text{good}, practice = \text{good}\}$, making its meaning more evident.

We also used this case to show the influence of β. As $\beta = 0.0$ was used, we can see that some branches represent explanations with a very low posterior probability (those in the dashed area in Fig. 6.4), and so they will not be useful. The dashed areas in Fig. 6.4 represent the parts of the tree that are not constructed if we use $\beta \simeq 0.05$, which apart of producing a simpler and more understandable tree is also of advantage to reduce the computational effort (probabilistic propagations) required to construct the tree.

With respect to the resulting trees for the gates case, we can appreciate two clear differences: (1) GINI produces simpler trees than MI, and (2) the most explicative variable is different depending on the used measure. Regarding this last situation, we can observe in the circuit that there are many independent causes[5] (faults) that

[5]However, it is interesting to observe that applying probability propagation, the posterior probability of each gate given the evidence, e.g. P(A1|x_O), indicates that that for all the gates it is more probable to be ok.

can account for the erroneous output. Choosing the AND gate A1 as GINI does is reasonable (as well as choosing A2) because AND gates have (in our network) greater a priori fault probability. On the other hand, choosing N2 as MI does is also reasonable (and perhaps closer to human behaviour) because its physical proximity to the wrong output. If we were a technician this would probably be the first gate to test. In this way, it seems that MI is more sensitive to the fact that the impact a node has in the value of the remaining nodes is attenuated with the distance in the graph.

Once the first variable has been decided, the algorithm tries to grow the branches until they constitute a good explanation. In some cases, it seems that some branches could be stopped early (i.e. once we know that N2=`fault`), but these situations depend on the thresholds used and it is clear that studying how to fix them is one of the major research lines for this work.

Perhaps an interesting point is to think about why O1 is not selected by MI when N2=`ok` as could be expected given the distance-based preference previously noticed. But, if we look carefully the circuit, we can see that output L (which is correct) also receives as input the output of gate O1, so it is quite probable that O1 is working properly.

Of course, we get different explanations depending on the used measure, the value of α or the criterion, but in general we can say that all the generated explanations are quite reasonable. Finally, in all the trees there is a branch, and so an explanation which indicates that a set of gates are ok. Perhaps this cannot be understood as an explanation to a fault, but we leave it in the tree in order to provide a full partitioning. Some advice about these explanations can be given to the user by indicating for example if such explanations raise or not the probability of the fault with respect to its prior probability.

6.4 Further experimentation

From the previous study and these simple examples we can extract that the explanations given by our *ET*-method are quite reasonable, as has been examined before. Anyway, this reasonability term might be a quite vague concept and we wish to reinforce the believe on the goodness of the generated explanation tree. Hence, we have figured out a more systematic way to contrast the obtained results to an already existing technique with the same purpose: K Most Probable Explanations search. In subsection 6.4.1 we will apply both techniques to the academe and gates problem in

order to compare the quality of explanations, and the differences in the format of giving them (always using global configurations or not). With the aim of going into greater detail, the next step will be a comparison with the already cited method using simplification of explanations[15]. To do so, we will use again the gates network and another of the networks employed in that work which models the start mechanism for a car (subsection 6.4.2).

6.4.1 *Explanation Tree* vs. Partial abduction

In this part of the chapter we intend to see the behaviour of our method and set the given solution against the K-best explanations generated by partial abduction. To be *fair*, and since the output of both methods is different, we will try to make a kind of translation from one to the other, in such a way that:

1. $ET \rightarrow K$-best explanations: From an Explanation Tree we will indicate the corresponding ranking of explanations, ordered by probability. In our method every leaf node represented one explanation, that one from the root until this leaf. Thus, this is almost immediate to be done, but that will make easier the search of similarities/differences when it is contrasted with the K-best explanations. With this, we can see if the ET is able of representing these K-best explanations and in which form.

2. K-best explanations $\rightarrow ET$: The K-best explanations provided by the abduction task will be reflected in a tree structure similar to the resulting ET. So we will annotate, how many explanations are included in each branch (and which ones). In this case we will measure the distributions of explanations along the tree, and see that, as expected, in partial abduction there are sets of configurations that could be aggregated in only one solution for the sake of simplicity (Principle of parsimony).

In this case, we do not deem necessary to go through all examples, covering the whole bunch of versions for ET provided before. We have decided to perform this illustrative proof with the most representative examples.

Academical example

For the academe network, we will look at the tree in figure 6.4.(a). We assume β-pruning has been performed, so the dotted area is removed as 6.7 shows. We have

chosen this tree (the largest one) on purpose, to avoid starting from an advantageous situation. To get the K-best explanations, we have fixed the K value to 20. We think it is a big enough number to guarantee that we allow partial abduction to have quite a lot explanations, more than the number of leaves, by far. In this academe problem MI and GINI happened to behave quite similarly, so we have not considered relevant to distinguish both cases.

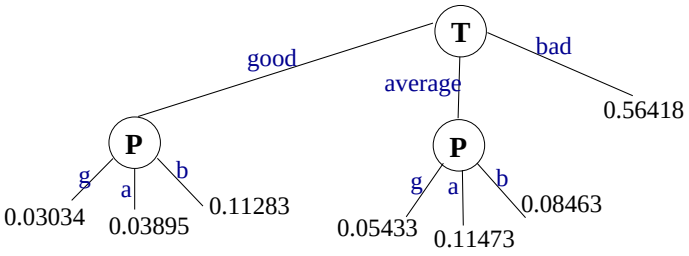


Figure 6.7: ET for academe and MI used for comparison and which is α,β-pruned.

In figure 6.8 we reproduce the output of running partial abduction (Elvira software) on academe network with $K = 20$, just for the same example we went through in the previous section. On the other hand, figure 6.9 has transformed the corresponding tree (fig. 6.7) into another ordered list of explanations to have the same arrangement style. In the second figure we have preferred not to add a column for every variable, since most of them will not be instantiated to a particular value.

First, if we just watch the first explanation in both cases, they coincide on the value of *Theory* which is equal to bad. This is a good sign in the sense that the most probable explanation is the same with the two techniques, but is this really exactly the same? Since the initial problem solving assumptions are different, these explanations are too. In MAP the first explanation is $\{Theory = \text{bad}, Practice = \text{good}, OtherFactors = +, Extra = \text{no}\}$. It seems quite logical that a student having a bad theoretical part had failed, but why should we assume he has done good practical assignments as this explanations says? That could lead to think that having bad practical assignments would not be a so nice explanation for this fact, and actually that would give even more reasons to think that the student has failed. The same happens with the *otherFactors* value, it is positive, but what if *otherFactors* would have been negative? Even worse, that will increase the possibilities of the student to fail the subject. That leads us to think that ET-explanations are more appropriate than K-MPE ones.

#	Theory	Practice	Other	Extra	Prob.
1	bad	good	+	no	**0.201776**
2	bad	avg	+	no	**0.084076**
3	avg	avg	+	no	**0.067259**
4	good	bad	+	no	**0.067259**
5	bad	good	+	yes	**0.064857**
6	bad	good	-	no	**0.053099**
7	avg	bad	+	no	**0.050044**
8	bad	bad	+	no	**0.050044**
9	bad	avg	+	yes	**0.027024**
10	avg	avg	+	yes	**0.027024**
11	bad	avg	-	no	**0.022124**
12	good	bad	+	yes	**0.021619**
13	good	good	-	no	**0.021240**
14	avg	good	+	no	**0.020178**
15	bad	good	-	yes	**0.019877**
16	avg	good	-	no	**0.019647**
17	avg	avg	-	no	**0.019027**
18	good	bad	-	no	**0.017699**
19	good	avg	+	no	**0.016815**
20	avg	bad	+	yes	**0.016214**

Figure 6.8: 20-best explanations for the academe model when $X_O = \{finalMark = failed\}$ and $X_E = \{theory, practice, otherFactors, Extra\}$.

#	Configuration	Prob.
1	$\{theory = \text{bad}\}$	**0.56418**
2	$\{theory = \text{average}, practice = \text{average}\}$	**0.11473**
3	$\{theory = \text{good}, practice = \text{bad}\}$	**0.11283**
4	$\{theory = \text{average}, practice = \text{bad}\}$	**0.08463**
5	$\{theory = \text{average}, practice = \text{good}\}$	**0.05433**
6	$\{theory = \text{good}, practice = \text{average}\}$	**0.03895**
7	$\{theory = \text{good}, practice = \text{good}\}$	**0.03034**

Figure 6.9: Explanation ranking corresponding to ET in figure 6.7.

Once the explanation content has been regarded, we should also look at the associated strength. In MAP the most probable explanation is 20% let us say *acceptable*. But in our ET the first explanation "*Theory* is bad" covers more than the half (56%) of the explanation space. We find this second number much more accurate: in most of the cases where a student has failed a bad theoretical exam seems a good enough explanation. If we observe the prior probability tables *Theory* was precisely the most influential factor. This confirms our believe that when abduction requires configurations always with $|X_E|$ elements, there are some unnecessary variables $X_{unnecessay} \subset X_E$ that change their states in the configurations in a counter-intuitive way (they are in fact being tuned), since this variable instantiation is not really significant for the explanation.

To understand better this important point, we jump to the other related illustration (translation K-best expl. $\rightarrow$ ET) in figure 6.10. Here we tried to make a picture on how the K-best explanations would be distributed in the ET. So, the structure is identical to the ET, i.e. a tree node is a variable, a branch indicates the state associated to this variable and leaf nodes represent explanations. Precisely at this lower level, where explanations are implicitly depicted, we have included the K-best explanations data:

- Together with the last branch configuration, and between brackets [ ], we annotate the explanations that would fall in that branch/explanation. The number indicates the ranking, as column # in figure 6.8.

- Just on the leaves a triangle indicates the number M of explanations that are included in this position (Mexpl.) and below it we write the sum of their probabilities, that is:

  $$\sum_{x_E \in Expl_{path}} P(x_E|x_O)$$

  where $Expl_{path}$ is the set of explanations in path going from root to the leaf and $|Expl_{path}| = M$.
  For example explanation #14: {*Theory*=avg, *Practice*=good, *OtherFactors*=+, *Extra*=yes} belongs to path <*Theory*=avg, *Practice*=good>.

- Finally, in braces we indicate the branch/path probability as usual in the ET representation.

Then, in figure 6.10 we can see how {*Theory* = bad} piles up the biggest number of explanations (8 expl.), even if we look at all the branches at the same detail level {*Theory* = good} has 5 and {*Theory* = avg } presents 7. However it is not so a

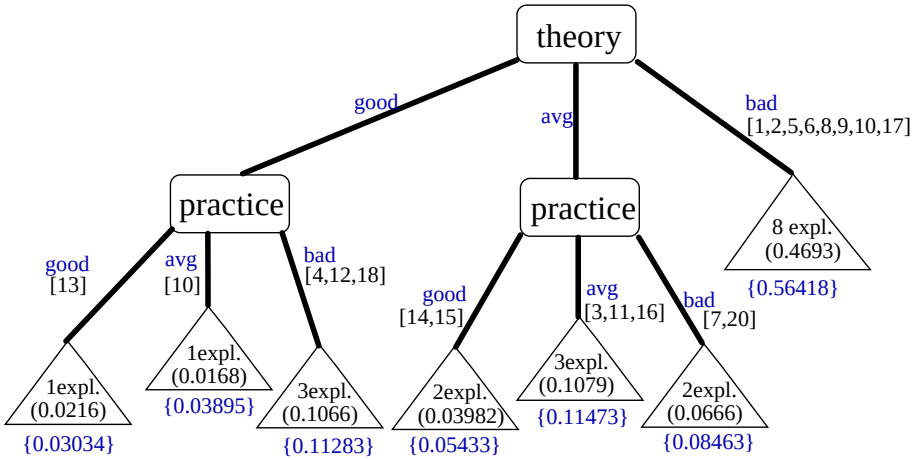


Figure 6.10: ET structure for academe including 20-best explanations.

question of quantity, but mainly *quality* and what is clear is that branch {*Theory* = bad} includes 7 of the 10-best explanations!!! Notice that in figure 6.8 if we apply a *similar* criterion to our β-pruning, with $\beta = 0.05$ we could have just taken the eighth first explanations, the other could be considered as too little relevant. Anyhow, as we have remarked above, we show them because we wanted to take a big *mass* of explanation in order to make the comparison advantageous to partial abduction in the way that many explanations are regarded. In a practical case, the user would not normally need to have up to 20 different reasons.

The main conclusion from the previous discussed point is that when we perform the search of the K-best explanations, they are not necessarily K. With the resulting ET we can see that for example 7 out of 10 explanations could be (and should be in our opinion) amalgamate in only one.

Even if we think that our method manages to outperform partial abduction in both efficiency and results, there are still doubtful issues in its own nature. For example, with the same principle as the previously explained, we could think that having *Theory* as average could also be an explanation by itself. Observe that ET-explanations #2,#4 and #5 presents that configuration and they differ in the practical part value. But here there could be nuances to take into account. For example having *theory* and *practice* ordinary (avg) is the second explanation, very near from having bad *practice* and good *theory* whereas average *theory* and good *practice* decrease considerably the probability when we know that the student has failed. This instability is even more clear when

theory is good, we need to distinguish the possible states for *Practice* to reach a valid explanation. Look at the different probabilities for these three branches: {*Practice* = bad} is 4 times more probable than the other two. Nevertheless, in the 20-best explanations this one would appear in the forth place without standing out specially from the rest, and later this explanation reappears in #12 and #18.

Circuit example

Unlike the first example, the provided ETs differ a lot from using MI or GINI as the *Info* measure both in size and also in the nodes situation along the tree. So, in this second case, we are going to divide this comparative study into two parts: one for tree in figure 6.5.(a) and the other for tree in fig. 6.6.(a). This last one is in fact the smallest tree, that is precisely why we have found of interest to examine this "minimal"[6] too from this juxtaposing perspective.

•When Info is Mutual Information

The notation for figures is exactly the same as for the previous example. So, we can find the 20-best explanations in fig. 6.11, the ranking for ET in fig. 6.13 and the *integrated* tree in fig. 6.14.

Looking at the two rankings, again we can detect that not only the first explanation, but the two first (which are equiprobable) are alike in the sense that the extracted anomaly is that either $A1$ does not work properly or the failure is in $A2$. But again, since the K-MPEs need a value for every variable, this method burdens this explanation with values that are not relevant, all the other gates are set to *ok*. Here the overspecification problem is again quite clear. And, from the quantitative point of view (probability ordering) the same situation repeats: as the significant data is that gate $A1$ (or $A2$) fails, the rest of configurations when $A1 = fault$ are regarded here, *losing* the corresponding probability for the other states. Luckily, in this example, this difference is slightly greater than 0.015 (0.32775 - 0.311406), because the accumulation of two gates faults is quite improbable. To check that, we should just look explanations from #5 to #20, where the probability barely reaches 0.007. Also, it is curious to see how three faults are not even considered within the 20 best explanations.

In *ET* depending on the side of the tree, there could be three or four gates included in the explanation. This gives us the hint that this designed algorithm is also able of discerning those variables within the explanation set which are in fact relevant to

[6] "Minimal" not in a strict way, just the minimum tree among the generated by our ET executions.

#	$N1$	$N2$	$A1$	$A2$	$A3$	$O1$	$O2$	Prob.
1	ok	ok	ok	fault	ok	ok	ok	**0.311406**
2	ok	ok	fault	ok	ok	ok	ok	**0.311406**
3	ok	ok	ok	ok	ok	fault	ok	**0.205486**
4	ok	fault	ok	ok	ok	ok	ok	**0.101705**
5	ok	ok	fault	fault	ok	ok	ok	**0.014447**
6	ok	ok	ok	fault	ok	fault	ok	**0.006355**
7	ok	ok	fault	ok	ok	fault	ok	**0.006355**
8	ok	ok	ok	fault	fault	ok	ok	**0.004815**
9	ok	ok	fault	ok	fault	ok	ok	**0.004815**
10	ok	ok	ok	fault	ok	ok	fault	**0.003178**
11	ok	ok	fault	ok	ok	ok	fault	**0.003178**
12	ok	ok	ok	ok	fault	fault	ok	**0.003178**
13	ok	fault	ok	fault	ok	ok	ok	**0.003145**
14	ok	fault	fault	ok	ok	ok	ok	**0.003145**
15	ok	ok	ok	ok	ok	fault	fault	**0.002097**
16	ok	fault	ok	ok	ok	ok	fault	**0.002076**
17	ok	fault	ok	ok	ok	fault	ok	**0.002076**
18	fault	ok	ok	fault	ok	ok	ok	**0.001573**
19	fault	ok	fault	ok	ok	ok	ok	**0.001573**
20	fault	ok	ok	ok	fault	ok	ok	**0.001573**

Figure 6.11: 20-best explanations for gates with $X_O = \{\texttt{ABCDEKL} = \texttt{0101000}\}$ and $X_E = \{A1, A2, A3, O1, O2, N1, N2\}$.

explain the given observations. $A1$ and $A2$ (with a symmetrical behaviour for this example) together with $N2$ are clearly the three involved gates that could have caused the given error. In this particular example, even without considering the threshold $\beta(=0.0)$, gates $O2$ and $A3$ never appear in the explanation tree. We can see that the erroneous output is in signal K and these two gates do not play a role for that value. In certain cases $O1$ (depending on the thresholds' values) could appear. As we already commented, this gate, even if it is related to the *bad* output K, increases its belief of working properly (or, the other way round, decreases its belief of being faulty) because it also participates in producing signal L, which is correct. In K-best explanations this happens to be the third possible faulty gate, so we find that this implication (L signal valid then $O1$ is likely to work properly) is not caught there.

The nice feature about ET selecting the explanatory variables has also been shown in the academe example where *Theory* and *Practice* stood out as the significant variables.

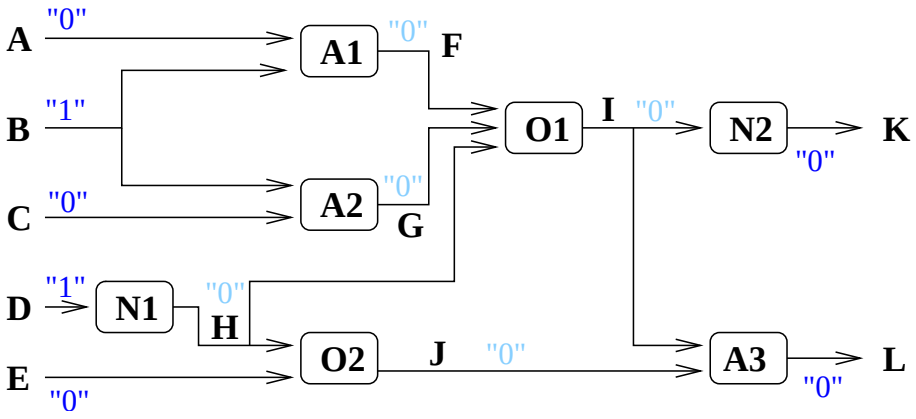


Figure 6.12: Circuit with the evidence (ABCDEKL=0101000) incorporated and the assumed signal values when if everything works ok annotated.

Also, let us comment that the set of *paired* explanations (for instance #1 and #2, #8 and #9, #10 and #11, $\cdots$) comes from the symmetrical influence of gates $A1$ and $A2$. From figure 6.12 this symmetry can easily observed.

A glance at the combined tree in fig. 6.14 reinforces the conjectures previously remarked. The two branches that accumulate a larger number of (MPE) explanations (5 of them each) are exactly the two first candidates: $A1$ is faulty or $A2$ is faulty. Next, unexpectedly maybe, we have that gates $N2$,$A1$,$A2$,$N1$ are ok. This explanation is

#	Configuration	Prob.
1	$\{N2 = \text{ok}, A1 = \text{ok}, A2 = \text{fault}\}$	**0.32775**
2	$\{N2 = \text{ok}, A1 = \text{fault}, A2 = \text{ok}\}$	**0.32775**
3	$\{N2 = \text{ok}, A1 = \text{ok}, A2 = \text{ok}, N1 = \text{ok}\}$	**0.21082**
4	$\{N2 = \text{fault}, A1 = \text{ok}, A2 = \text{ok}\}$	**0.10809**
5	$\{N2 = \text{ok}, A1 = \text{fault}, A2 = \text{fault}\}$	**0.01510**
6	$\{N2 = \text{ok}, A1 = \text{ok}, A2 = \text{ok}, N1 = \text{fault}\}$	**0.00373**
7	$\{N2 = \text{fault}, A1 = \text{fault}\}$	**0.00343**
8	$\{N2 = \text{fault}, A1 = \text{ok}, A2 = \text{fault}\}$	**0.00333**

Figure 6.13: Explanation ranking corresponding to ET in figure 6.5.(a).

necessary to be given, since our second concern was to give a partition of the explanation space, but for these cases we plan to use the explanatory power (EP) defined in [22] in order to give them a second level of classification on this value. This value is obtained by $EP(X, E) = \frac{P(E|X)}{P(E)}$, being X an explanation (next variable) and E the observations (X_O plus *path*). Using this concept, in this case both $O1$ and $O2$ would have failed this *test*.

Again, notice that if we had taken for example the 10-best explanations, our tree would have given even a deeper level of detail just with 8 leaves/explanations because some leaves will not correspond to any of the 10-best explanations, ET adds then more information.

Finally, we would like to add again that the small probabilities of explanations from #5 to #20 prove that they were unnecessary. Even though, with a few propagation steps with the explanation tree we are able of capturing all of them. Besides, these explanations are presented to the user in a more intuitive and simpler manner.

•When Info is GINI index

Using GINI index instead of MI has changed the order of selected variables in the tree, as reviewed in section 6.3. Also, see that in figure 6.6.(b) the second level variable is distinct depending on the state taken by the root variable $A1$. But we are going to examine tree 6.6.(a) and when incorporating the 20-best explanations on it we obtain the one depicted in fig. 6.16. As this tree shows and as the corresponding ranking (fig. 6.15) does too, again the two main explanations are that either $A1$ or $A2$ does not work. In this case $A1$ has been first selected, but we think is a question of tie breaks.

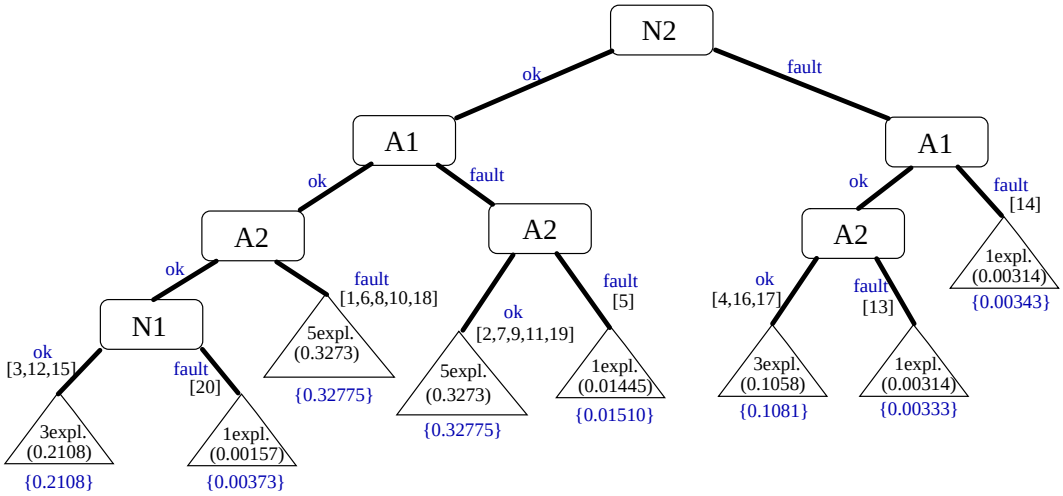


Figure 6.14: ET structure for gates with MI including 20-best explanations.

Apart from the tree level of the involved gates and the slight difference between ET-explanations #1 and #2 the same conditions as with MI accomplish. But this time we have the advantage of presenting a more compact tree, where the system determines that the relevant gates to be tested are $A1$, $A2$ and $N2$ in this order (according to the probability ranking). This is influenced by the thresholds α and β [7], as the other technique was influenced by the K number. The user can make a choice on these values depending the level of detail s/he requests. But for a general case these explanations seem good enough and they have been reached with a reasonably low effort, just a few propagations on the join tree. Once again, we can also say, that the method has made a reduction of the explanation set from seven variables down to only three, which also simplifies quite a lot the problem.

6.4.2 Looking into simplification methods. A new example: *car-start*

To finish in this evaluation of the *Explanation Tree* technique, we find interesting to add some detail about the already mentioned method of simplifying explanations [46, 15]. We just would like to set out the basic ideas in order to see similarities and differences of this method with our proposal. In these two works, the objective

[7] And by the min, avg or max criteria as well.

#	Configuration	Prob.
1	$\{A1 = \text{fault}\}$	**0.34628**
2	$\{A1 = \text{ok}, N2 = \text{ok}, A2 = \text{fault}\}$	**0.32775**
3	$\{A1 = \text{ok}, N2 = \text{ok}, A2 = \text{ok}\}$	**0.21456**
4	$\{A1 = \text{ok}, N2 = \text{fault}, A2 = \text{ok}\}$	**0.11141**

Figure 6.15: Explanation ranking corresponding to ET in figure 6.6.(a).

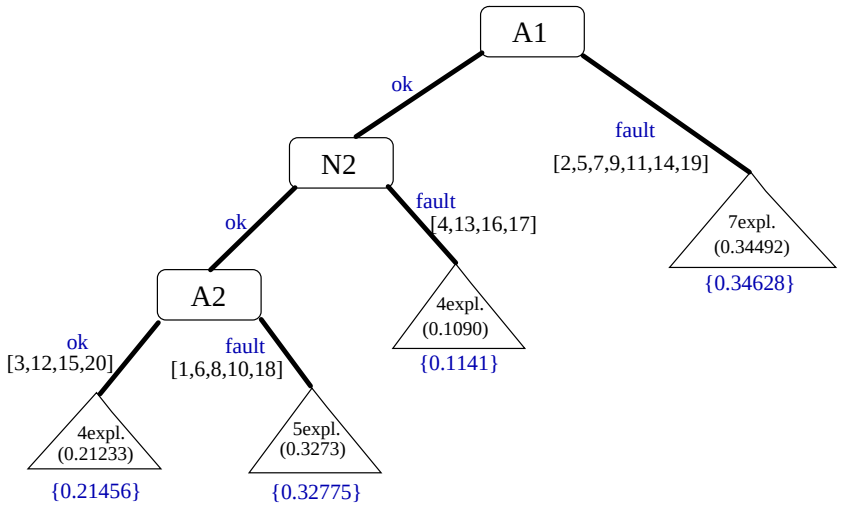


Figure 6.16: ET structure for gates with GINI including 20-best explanations.

was to simplify the explanations given after applying MPE as figure 6.17 illustrates. They also state the process as follows: "*Let* $expl(x_O) = \{x_E^1, x_E^1, \ldots, x_E^k\}$ *be the* K *MPEs obtained for evidence* $X_O = x_O$. *Then, for all* $x_E \in expl(x_O)$ *we are looking for a sub-configuration* x'_E $[X'_E \subset X_E]$, *so that* x'_E *is still accounting for the observed evidence.*"

So, the process is differentiated into two main steps:

1. Generation of complete explanations (configurations of X_E with $|X_E|$ literals), ordered by their posterior probabilities given the observations.

2. Simplification of these explanations by removing *unimportant* literals.

To design a mechanism for this *reduction* of explanations, the authors define and propose two important criteria: **Independence** (I∼simplification) and **Relevance**

Figure 6.17: Process followed in the simplification of explanations in [15].

(R∼simplification)[8]. We avoid theoretical details and formulas, aiming the conception of both criteria. The first one will try to detect those variables that are useless in the explanation since the value this variable takes does not affect the evidence. In the case of relevance-based criteria they attempted to remove those literals that could be considered as insignificant in the sense that they are (almost) irrelevant for the observed evidence.

We see easily an analogy between these two kind of simplification *rules* and our two methods for pruning the explanation tree. α value was related to the information measure and intended to determine when a variable is not decisive to be in the tree because it does not contribute with new information. Mutual information of independent variables is zero, so the threshold α measures in some extent the independence factor. On the other hand, with β-pruning we wanted to avoid explanations with a very little probability associated, estimating them as trifling, that is irrelevant, which resembles the second criteria. However, there is a notable difference between both techniques: in ET the root variable will always be in all the explanations whereas when simplifying explanations it could happen that one explanation is $\{A = a_1, B = b_1\}$ and another one $\{C = c_1, D = d_1\}$.

It is clear that both approaches come up from the same concern: avoid the overspecification problem when dealing with abductive inference in BNs. In fact, the current work was inspired on the previous one, but attempting to skip its two main drawbacks: (1)little efficiency.- since it requires a two-step process and the first one includes K-MPEs and (2)being this simplification *K-MPE guided* the obtained solutions are somehow influenced by them. As we have just verified, in many occasions the K-best explanations do not correspond to K different scenarios, because some explanations should have been aggregated into one. Remember the previous example, where 7 of the 10th best explanations could be summarised in only one. We observe that these 7 explanations are just the combination of the simple explanation extended with different

[8]In this work the authors develop other simplification techniques such as those induced by the graph, that we will not touch on here.

configurations of other variables. We also believe this happens very frequently when performing partial abduction. Actually, the experiments executed in [46] discovered that after applying simplification to the K-best explanations, most of the times the same subset of simplified explanations were repeated following a certain pattern.

To confirm our feeling that ET succeeds not only in giving explanations with different number of literals, but also in improving the simplification procedure in [15] we are going to take a couple of examples in this work and apply the ET-algorithm on them.

To start with, we take again the gates network, but this time we fixed the evidence to $X_O = \{D = 0, K = 1, L = 1\}$ and the explanation set to $X_E = \{F, G, H, I, J, A1, A2, A3, O1, O2, N1, N2\}$. To have a visualisation of this situation we have depicted fig. 6.18. They have selected the explanation set as those variables that are not observable in the network.

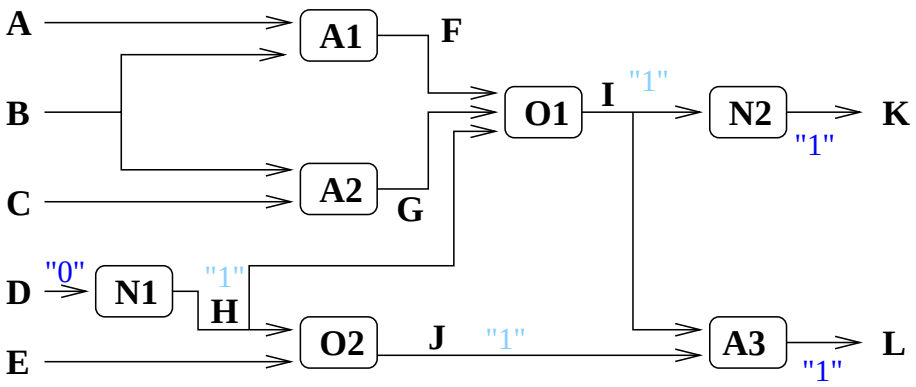


Figure 6.18: Circuit with the evidence (DKL=011) incorporated and the assumed signal values when everything works ok annotated. Notice that here there is a conflict: if output L is 1, and being $H = 1$ an input of the OR-gate $O1$, then I should be 1, but that will imply $K = 0$ which is inconsistent with the evidence.

For this example in [46] it is applied a technique called successive explanations search that after several stages gives rise to explanation $X_S^5 = \{A3 = ok\}$.

If we execute the ET-algorithm on this same data and some of the *standard* thresholds (for example $\alpha = 0.05, \beta = 0.05$) with MI the given tree is as simple as the one depicted in figure 6.19.

Well, we could say that both explanations are not inconsistent, $A3$ can work prop-

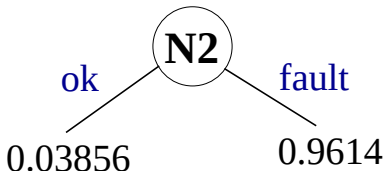


Figure 6.19: ET obtained for the gates with (MI,0.05,max) and $\beta = 0.05$ in the situation of fig. 6.18.

erly at the same time as $N2$ presents a fault. Besides, if $A3$ works a failure in $N2$ is the fact that better would explain this circumstance, because the output of $A3$ should be correct. Thus, if $L = 1$ that means that $I = 1$, but this is an input for the NOT-gate $N2$ that should have been the opposite signal, 0, according to the introduced evidence. We think that the second explanation ($\{N2 = fault\}$ with 96% of strength) is quite more informative in a diagnosis task. So we really find that our answer is of better quality than the other one.

Another example: car-start problem

As a second example we have taken the car-start network from [15][9]. We are going to firstly introduce a simple and intuitive example, where the evidence is that the car does not start (*Starts* = No), and the explanation set is $\{X_E = Alternator,$ *FanBelt, Leak, Charge, BatteryAge, BatteryState, BatteryPower, GasInTank, Starter, EngineCranks, Leak2, FuelPump, Distributor, SparkPlugs*$\}$. With an ET-execution of kind [Info,criterion,α,β] =[MI,min,0.07,0.05] the resulting tree is the one in fig. 6.21.

Even with real probability tables unknown, any person with a little knowledge about cars could try to interpret this answer which just says that the two most probable explanations are that the battery state is weak (0.77) or as a second and less probable explanation (0.13) that the started could be faulted.

Let us now show a more complex case studied in [46] where evidence is $X_O = \{GasGauge =$ Gas, *Lights* = Work, *Radio* = Works, *Starts* = No$\}$ and the explanation set is the same of the previous example. With the simplification method the final obtained explanation is $\{GasInTank =$ Yes, *Starter* = Faulted$\}$. We have performed an execution of the Explanation Tree algorithm of kind (MI,min,0.07,0.05) as before,

[9]They indicate that this network has been originally found in JavaBayes package. http://www.cs.cmu.edu/∼javabayes is the web site.

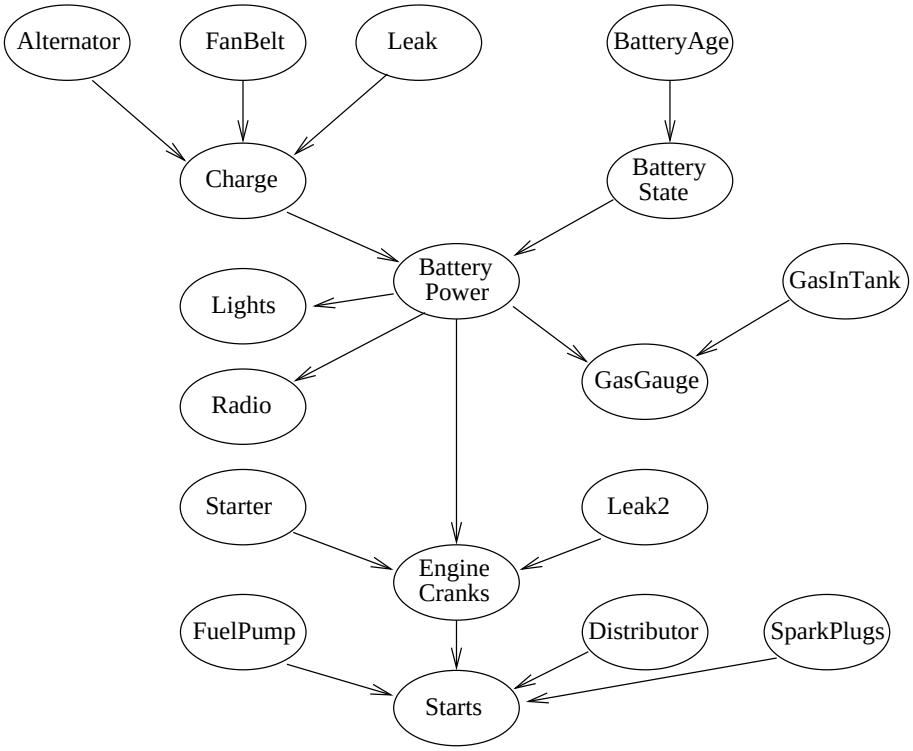


Figure 6.20: Network modelling the car-start problem.

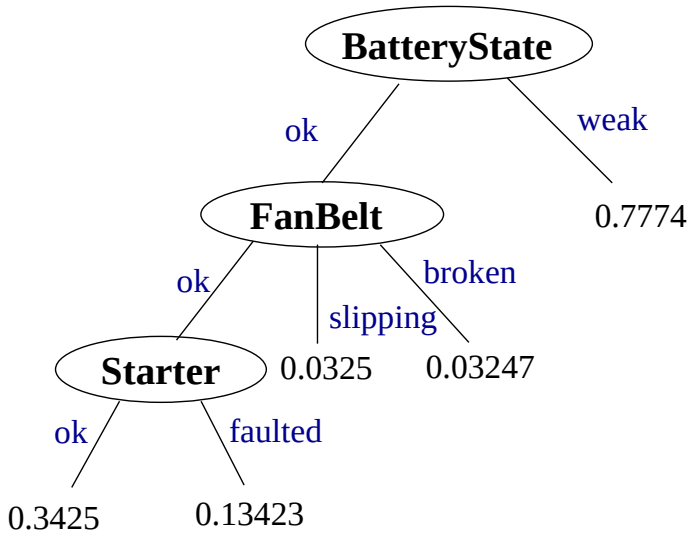


Figure 6.21: ET for the car-start problem, when $\{Starts = \text{No}\}$ and (MI,min,0.07,0.05).

and the given tree is drawn in fig. 6.22. In this case we find that both simplified explanations are quite close. It is clear that the starter does not work properly, which is probably the main explanation. But both simplification and ET go further adding also that there is no problem with the gas status. In simplification it says that there is gas in the tank while ET has *checked* the possibility of presenting a leak of kind "2", but this is quite improbable. Since the evidence included that the gas gauge indicates enough gas, and this device is not determined as faulted, we could have supposed that it works properly and therefore $\{GasInTank = \text{Yes}\}$.

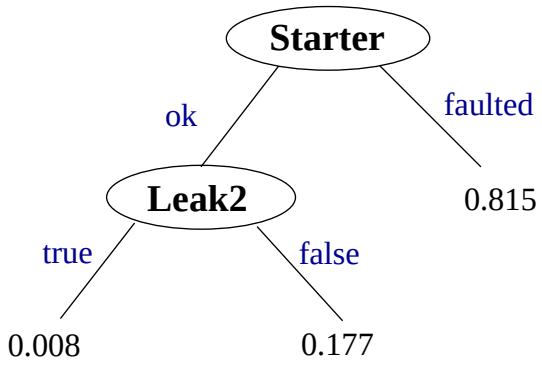


Figure 6.22: ET for the car-start problem, when $\{GasGauge = \text{Gas}, Lights = \text{Work}, Radio = \text{Works}, Starts = \text{No}\}$ and (MI,min,0.07,0.05).

We have performed a final comparison "*simplification vs. ET*" taking randomly selected examples from [15]. In this study they just indicate the explanation set and the observed variables and, using different types of algorithms for simplification, they show the obtained number of literals. We have then reproduced these tests with all the possible configurations for the observed variables with (MI,max,0.07,0.05) and in all cases either the tree was empty or it presented one variable. In simplification experiments the number of literals could vary from 1.5 until 6. In the cases where an empty tree was obtained, it could indicate that all variables have a very low entropy and they present a fixed value. Then, there is only one possible scenario that will be the most probable explanation. On the other hand, in the cases with one variable the distribution was quite clear giving probability numbers such as 0.933 or 0.8146.

6.5 Discussion and further work

This chapter has proposed a new approach to the problem of obtaining the most probable explanations given a set of observations in a Bayesian network. The method provides a set of possibilities ordered by their probabilities. The main novelties are three:

1. The level of detail of each one of the explanations is not uniform (with the idea of being as simple as possible in each case).

2. The generated explanations are mutually exclusive.

3. The number of required explanations is not fixed (it depends on the particular case we are solving).

Our goals are achieved by means of the construction of the so called *explanation tree* which can have asymmetric branching and that will determine the different possibilities.

We have described the procedure for its computation based on information theoretic criteria. To show its behaviour some simple examples have been proved and we have presented a comparison with the K-best explanations and with the simplification of explanations. From this experimental analysis, we can conclude that our method outperforms the K-best explanations search by far both in quality and efficiency, since our method minimises the number of propagations and manage to do them in a very quick form. We find that the description given by an ET is more useful for a user than the one given by k complete explanations because in the former case the possibilities are given in a more compact and structured way.

With respect to efficiency we have that ET construction is polynomic in the number of cases where probabilistic propagation is polynomic in the case of polytrees, whereas K-MPEs are NP-hard in these cases.

Regarding the simplification of explanations, we could say that the obtained explanations are different, but not always of different quality. Nevertheless, *ET* is able of giving simpler explanations and what is more important it achieves the task in a direct way, avoiding the cost of performing *traditional* abduction first.

Our main conclusion is that this technique behaves in a very satisfactory way, but we would like to run more sophisticated experiments in order to evaluate it in depth. We believe it is quite promising, even though we are conscious that a better refinement should be done for some issues such as the determination of α and β thresholds or the

characterisation of those explanations given for partitioning the explanation space, but not relevant as an explanation.

As further work we plan also to study the possibility of using Kullback-Leibler distance as the Info measure. In addition, very useful comments have suggested us to explore the connection about this method and the so-called Hitting set which is quite broadly studied by researchers on the logical field together with theories of diagnosis [104]. Finally, we would like to analyse if our diagnosis method could be benefitted from the conflict analysis theory, as long as conflicts are applicable to the particular problem.

Conclusions and further work

Conclusions

As the main title says, *Bayesian networks inference: Advanced algorithms for triangulation and partial abduction*, the global goal of this work was to do a thorough study of inference performance in Bayesian networks, and the investigation of advanced algorithms to optimise this (normally) complex process.

We have undertaken this problem from three distinct, but related, aspects:

1. Improvement on the compilation process that constructs a join tree from a Bayesian network.

   Chapter 2 have dealt with the most problematic step within compilation: triangulating the graph. We have presented a new method for triangulation which makes use of a decomposition of the graph in order to subdivide the problem into smaller triangulation problems (Re-Triangulation). The presented results prove that the process of triangulation can generally be benefitted from our technique. This contribution was also published in [42].

   Chapter 3 goes one step further trying to tackle the whole problem of compilation (triangulation + tree construction) in a more efficient way. We have presented the MPSD-based Incremental Compilation technique. This new algorithm is able of avoiding unnecessary recompilation for a network that has been only partially changed. The shown experimentation, made through a set of networks, confirms that the behaviour of the Incremental Compilation is quite good, especially for large networks and a few modifications. This scenery is precisely the expected one, since IC is particularly thought to make easier and quicker the process of compiling a network during the *refinement* steps, and no big changes are supposed to be done at once. Since the complexity of compilation grows with the complexity of the networks, an important result is that the gain of this

technique (*versus* traditional full compilation) increases for large networks. These contributions can be found in two publications: algorithmically in [43] and an initial empirical study in [44].

2. Improvement on the inference process when working with modular structures, in particular with Object Oriented Bayesian Networks.

   Even if chapter 4 is a revision on already existing methods, it has involved a hard task of documentation and assimilation of the main notions and complex algorithms related to them. This previous study was necessary to develop the Plug & Play inference technique and the reported summary gives the required new information in order to understand its description in chapter 5. The main contribution of Plug & Play (published in [6]) is the design of an efficient method to perform inference in OOBNs, that is, another advanced algorithm for inference.

3. Improvement on the abductive inference for Bayesian networks.

   The search of explanations is clearly another important research field which implies different inference techniques. Partial abduction has been more studied in the recent years. Nevertheless, the lack of conciseness and the fixed number of variables in the given explanations make the solution less satisfactory than a real user would expect. We have succeeded in the design of a new algorithm that resolves these two problems, and it is at the same time able of presenting a partition of the explanation space. We have called this new abduction method as Explanation Tree since the given explanations are presented in a form of a tree. This technique was first published in [45], and in this work we have extended the study of its behaviour through a set of new examples in order to compare it with other existing techniques. This comparative gives nice results about the new technique, although we plan a more sophisticated experimentation in the future.

Further work

Through the different chapters of this thesis report some new proposals and ideas have already been launched. The main ones will be summarised here.

Regarding triangulation, we would like to utilise the concept of α-completeness. By that, we could design an α-ReTriangulation method, that is, a heuristic version

of our original algorithm, but with the advantage that the division of the graph in α-MPSs can be more profitable for the *Divide & Conquer* philosophy of this technique. The method will not be deterministic, i.e. the division will depend on factors such as the value of α or the way of splitting groups. So, after defining the technique, we plan to do a serious experimentation to study its behaviour.

Another idea we would like to implement is the obtaining of elimination orderings guided by the join tree structure, or by the α-Maximal prime subgraph tree.

This concept links directly to another commented approach: a possible future adaptation of MSBN inference communication in the IC technique. Again, we would like to utilise the concept of α-completeness to make a more balanced division of the problem. Since when subgraphs are not prime the original algorithm we developed is not correct, we believe that the fill-in propagation scheme might be applied to accomplish this task.

With respect to the Incremental Compilation, we find that it can be successfully applied to certain situations, and that a deeper study with real users can be very interesting. We also intend to enhance the implementation in order to make it even more efficient. An analysis of the applicability of IC to other graphical structures beyond BNs and OOBNs can be of great interest too.

Finally, we think there is still an open field concerning abduction. The *Explanation Tree* technique seems to give nice and promising results, but we feel it should be studied and improved in a short future. As another immediate and important goal we also expect to use this technique in the area of data clustering. Since this algorithm is able of classifying instantiated variables in mutually exclusive subsets, we reckon that it can probably be adjusted to work with data bases in order to detect relevant configurations of variables.

Appendix A

Aspects of implementation

In this appendix we will deal with certain points about the implementations carried out along the whole work. We intend to indicate and remark the main points, without going into minor details. The three implementation tasks included in this thesis are those for which we have shown any experimentation results:

1. Triangulation by re-triangulation (chapter 2).
2. MPSD-based Incremental Compilation (chapter 3).
3. Explanation Tree (chapter 6).

In the next three sections we will tackle these three items with the aim of giving the most important features. These three tasks have needed a considerable amount of time and work to obtain a definitive structure for the different techniques in classes and methods as well as the design, validation and performance of the corresponding experiments. However, we would like to remark that the Incremental Compilation method has definitively involved the greatest effort from the programming point of view. This is due to the difficulty of replacing the *traditional* compilation method, which obtains a join tree from the network in a direct way, by partial compilations that should extract the affected parts in both structures and then recombine the obtained trees. These particular troubles belonging to the nature of the method itself had first to be solved algorithmically (chapter 3), but their translation into programming code was not an easy task either. That is why we will devote more space and some diagrams to show the main perspectives. We have considered suitable to clarify the main implementation aspects for this method (section A.2) using the standard UML for diagrams. We assume that the reader will be familiarised with this notation. Anyway, all of them will be commented and described along the text.

A.1 Triangulation by re-triangulation

This is not only the first technique described in the report, but also the first one we implemented. When we programmed this method we decided to make a *reduced* version of the Elvira code in order to focus uniquely on the graphical triangulation process. That is, we intended to avoid working with further information (unnecessary for our concrete initial goal) which makes Elvira code so powerful but also slower and mainly difficult to work with in this preliminary approach to the code. The programming language we chose is also JAVA, and we created simplified and adapted classes of Graph and Bnet. We also had to add new classes for the MPS Decomposition (MPSDTree and MPSDGraph).

Initially, we had to program the method to obtain the MPS Tree (alg. 11 described in detail in [93]). With all this, we could then include our method RE-TRIANGULATION technique and finally a test suite was designed in order to execute it on different networks and obtaining several statistics and data.

A.2 MPSD-based Incremental Compilation

For this particular task, we decided to integrate it completely in the Elvira Project software [39]. Downloads of the current version or a deeper view in the whole classes structure can be done through the website `http://leo.ugr.es/~elvira`. In this section, we will focus on the classes specifically created for our purposes, even though some new methods have certainly been needed in those classes representing the basic, but not simple, elements such as graph, Bayesian network, Join Tree, etc...

We would like to indicate that we had to implement a compilation method that followed the steps described in figure 3.3 because the construction of the MPS Tree needed the condition of having a minimal triangulation, and that forced to a method able of controlling such circumstance.

Once this *extension* had been done, we could then start the task of implementing the Incremental Compilation method. The final result can be summarised as figure A.1 illustrates. As alg. 15 showed there are two main steps in the Incremental Compilation method:

1. The performance of the modification(s)[1]: RUNICMODIFICATION.

[1]Remember that IC could receive a single modification or a list of them.

2. The connection of the obtained tree(s) with the unchanged parts of the original one: MAKECONNECTION.

When implementing this class these two main methods appear as well. The first one, RUNICMODIFICATION, can be executed several times depending on the number and nature (deleting a node involves the previous deletion of any incident link) of the modifications to be done. The execution of a modification is divided into two stages: modification of the graph and marking of the affected maximal prime subgraphs. To perform these actions, the main class IncrementalCompilation will be assisted by the abstract class ICModification, as figure A.1 shows in the first stage of the process. In this part, the JoinTree class plays an active role as well, since the affected clusters are taken from it. We can also observe the need of having accessible the trees. We should note that, since they share the same structure, both MPS Tree and Join Tree belong to the same class, differing only in tha boolean value of one attribute (isMPS).

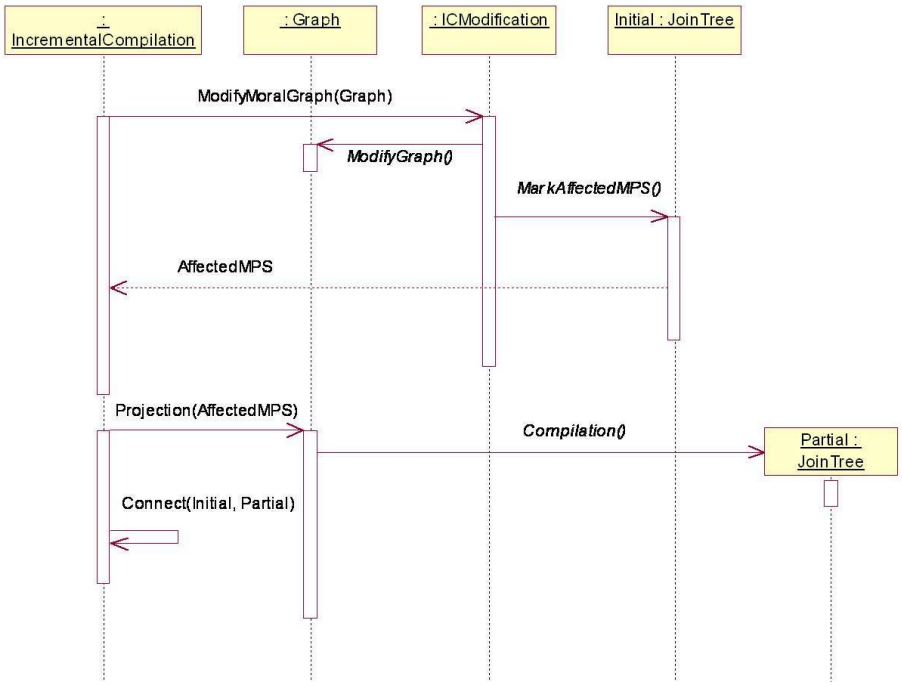


Figure A.1: Sequence diagram which describes classes interaction including the main executed methods chronologically.

Another important implementation feature is the diversification of the abstract class ICModification into four different classes, as figure A.2 depicts. This inheritance

solves the different actions to be performed according to the kind of modification, as several *switch* sentences indicated in the algorithms of chapter 3. From fig. A.2 we can also see the simplification of the operations for removing (alg. 19) or adding a node (alg. 20) which present fewer methods in contrast to the more complex classes for the modifications of addition/deletion of one link (algs. 21/18). As discussed in the corresponding chapter this situation comes from our particular approach when performing the distinct modifications.

Again taking fig. A.1 we can jump to the second important part (MAKECONNECTION). In this sequence diagram we show how the IncrementalCompilation class, given the affected MPSs is able of obtaining the projected subgraph(s). This/these subgraph(s) will be compiled in a similar way as the original and complete one producing (a) partial join tree(s). Finally, the main class will complete the action of connecting the initial tree with the partial one(s). This operation is probably one of the most complex ones in both the algorithm and the code, since many exceptional situations (for instance empty connections) had to be taken into account, as well as many structures had to be controlled and *synchronised* (clusters, trees, ..). So, we find this *connection* action a key issue for both design and implementation of our technique.

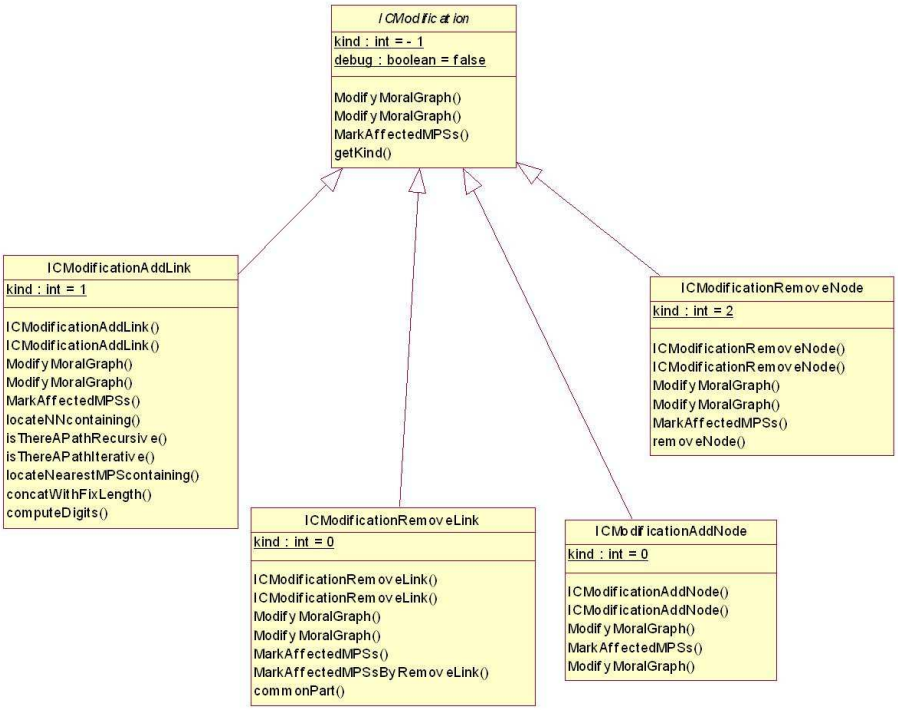


Figure A.2: Abstract class ICModification and its four inherited classes.

With figure A.3 we attempt to show the interaction between the main class IncrementalCompilation with both the modifications and with the needed structures: moral graph, directed graph and trees. We already mentioned when explaining algorithms in chapter 3 that we assumed to have them accessible, since they are used along all the process.

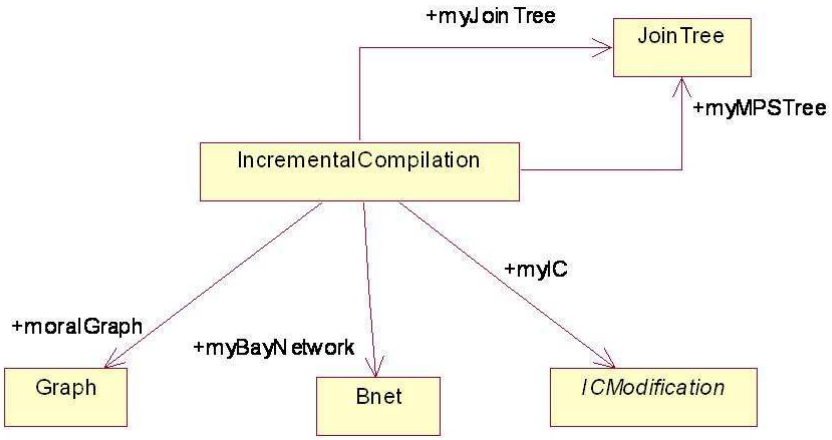


Figure A.3: IncrementalCompilation class and all the relevant elements it has to access in order to perform its actions on the network, graph and cluster trees.

In fig. A.4 all the attributes and methods for the class IncrementalCompilation are shown. Among them, we can find the previously commented major methods/actions: runICModification (or its *batch* version runListOfModifications that will invoke various times the other) and makeConnection[2]. The item labelled as myIC indicates the attribute/field that the modifications maintain to refer the main class.

A.3 Explanation Tree

This abduction technique has also been integrated in the Elvira Project code, although we plan to do further refinements on it. We also would like to design a systematic way of performing an experimental suite through a set of networks.

At this moment, there is a main class called ExplanationTree which interacts mainly with ShenoyShaferPropagation because our main method performs several propagations

[2]When a method appear more than one is due to the *overload* option possible in JAVA.

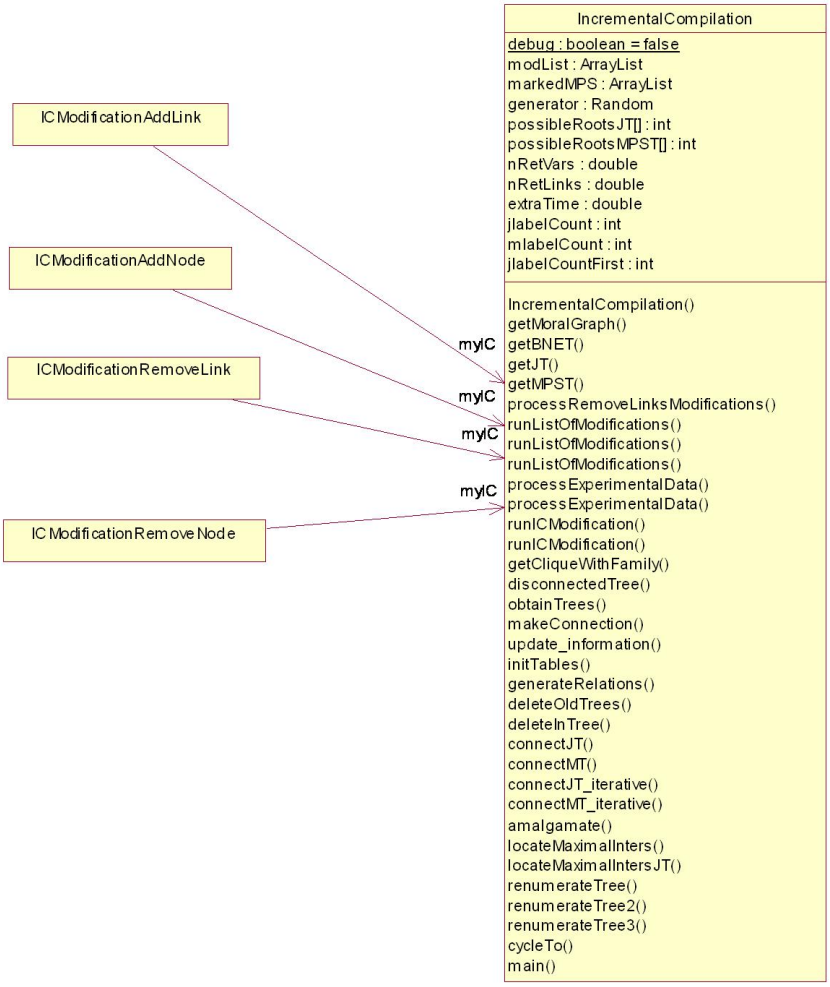


Figure A.4: Main class IncrementalCompilation associated to the different modifications ICModificationXXX and showing all methods and attributes.

over the join tree. As indicated in chapter 6 certain optimisations were performed on the original Shenoy-Shafer algorithm and we implemented in this class a particular adaptation of this architecture with the aim of accomplish our specific purposes. These particular new improvements can be summarised in these main points:

- For every variable in the network, the binary join tree will present a leaf cluster containing only this variable (singleCliques). We will use this in order to introduce/retract evidence. In our method this was done incrementally, the creation of a new branch/path will imply adding a simple configuration $X = x_i$ to the

evidence. The same happens when a branch is stopped or pruned, the evidence elements are retracted one by one.

- For every family, there exist a cluster containing its corresponding potential, and this cluster will uniquely contain this one. This feature will make easier the computation of marginal probabilities (for that they were called marginalCliques).

- We have used probability trees to represent the probability distributions of the potentials. This usually makes computations faster. They are also nicer and quicker for approximate methods.

- Special methods to enter/retract one evidence on a single variable (EvidenceItem) has been implemented. Thanks to the tree structure, the procedure results simpler and some computations can be saved.

- For certain tasks, we have used HashTables which contribute to a quicker access to the needed elements (variables, clusters, etc...).

On the other hand, in order to represent the tree we have made use of an already existing structure: the ProbabilityTree. This class provides a probability distribution in the form of a tree used to represent potentials (PotentialTree). For example, potentials can also be described by means of a PotentialTable (in the form of a table). The reason why we chose this structure for our ET is that probability trees can be represented in a simplified mode when all hanging branches present the same probability. So, we could construct explanation trees of different level of complexity (or different depth).

Therefore, in our case we could reuse all its information interpreting it as an ET where every branch will represent a configuration, and the leaf value will indicate the probability associated to this branch/path. The method CreateNewNode which can be found in the class ExplanationTree has been implemented recursively as in alg. 26. This method performs all the necessary calculations and verifications stated in the algorithm making use of auxiliary calls and is the one charged of the accumulation of nodes in the returned tree.

Bibliography

[1] S. Acid, L.M. de Campos, A. González, R. Molina and N. Pérez de la Blanca. CASTLE: a tool for Bayesian learning. In: *Proceedings of the ESPRIT 91 Conference, Commission of the European Communities*, pp. 363–377, 1991.

[2] E. Amir. Efficient approximation for triangulation of minimum treewidth. In:*Proceedings of the 17th Conference on Uncertainty in Artificial Intelligene (UAI-01)*, pp. 7–15. 2001.

[3] S. Arnborg, D. G. Corneil and A. Proskurowski. Complexity of finding embedding in k-tree. *SIAM Journal of Algebraic and Discrete Methods* **8**(2): 277–284,1987.

[4] O. Bangsø and P.-H. Wuillemin. Object oriented Bayesian networks, a framework for top-down specification of large Bayesian networks with repetitive structures. Technical report, Department of Computer Science, Aalborg University, Denmark, 2000.

[5] O. Bangsø and P.-H. Wuillemin. Top-down construction and repetitive structures representation in Bayesian networks. In:*Proceedings of the 13th International Florida Artificial Intelligene Research Society Conference (FLAIRS-2000)*, pp. 282–286, AAAI Press, 2000.

[6] O. Bangsø, M. J. Flores and F. V. Jensen. Plug & Play Object Oriented Bayesian Networks. In:*Current Topics in Artificial Intelligence, CAEPIA-TTIA 2003*, Lecture Notes in Artificial Intelligence, volume 3040, pp. 457-467, Springer Verlag, 2004.

[7] O. Bangsø. Object Oriented Bayesian Networks. PhD thesis, Computer Science Department, Aalborg University, 2004.

[8] V.E. Barker, D.E. O'Connor, J. Bachant and E. Soloway. Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM*, **30**(3):298–318, ACM Press, USA, 1989.

[9] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In: *Proceedings of the 12h Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pp. 81–89, 1996.

[10] A. Berry, J.R.S. Blair, P.Heggernes and B.W. Peyton. Maximum Cardinality Search for Computing Minimal Triangulations of Graphs. *Algorithmica* **39**(4): 287–298, 2004.

[11] A. Berry, J-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. To appear in *Journal of Algorithms*.

[12] A.Berry, J.R.S. Blair and P. Heggernes. Maximum Cardinality Search for Computing Minimal Triangulations. *WG '02: Revised Papers from the 28th International Workshop on Graph-Theoretic Concepts in Computer Science.* In: Lecture Notes in Computer Science, Vol. 2573, pp. 1–12, Springer Verlag, 2002.

[13] J.R.S. Blair, P.Heggernes and J.A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science* **205**(1-2):125–141, 2001.

[14] H.L. Bodlaender, A.M. Koster, F. van den Eijkhof and L.C. van der Gaag. Preprocessing for triangulation of probabilistic networks. In: *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligene (UAI-01)*, pp. 32–39, 2001.

[15] L.M. de Campos, J.A. Gámez, and S. Moral. Simplifying explanations in Bayesian belief networks. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, **9**:461–489, 2001.

[16] L.M. de Campos, J.A. Gámez, and S. Moral. On the problem of performing exact partial abductive inference in Bayesian belief networks using junction trees. In: B. Bouchon-Meunier , J. Gutierrez, L. Magdalena, and R.R. Yager, editors, *Technologies for Constructing Intelligent Systems 2: Tools*, pp. 289–302. Springer Verlag, 2002.

[17] L.M. de Campos, J.A. Gámez, and S. Moral. Partial abductive inference in Bayesian networks by using probability trees. In: O. Camp, J. Filipe, S. Hammoudi, and M. Piattini, editors, *Enterprise Information Systems V*, pp. 146–154. Kluwer Academic Publishers, 2004.

[18] A. Cano and S. Moral. Heuristic algorithms for the triangulation of graphs. In: *Proceedings of the 5th International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU), Vol. 1*, pp. 166–171, Paris (France), 1994.

[19] A. Cano, S. Moral, and A. Salmerón. Penniless propagation. *International Journal of Intelligent Systems*, **15**:1027–1059, 2000.

[20] A. Cano, S. Moral, A. Salmerón. Lazy evaluation in Penniless propagation over join trees. *Networks*, **39**:175–185, 2002.

[21] E. Castillo, J.M. Gutiérrez and A.S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer-Verlag, 1997.

[22] U. Chajewska and J. Y. Halpern. Defining explanation in probabilistic systems. In: *Proc. of 13th Conf. on Uncertainty in Artificial Intelligence (UAI–97)*, pp. 62–71, 1997.

[23] P. Cohen and M. Grinberg. A theory of heuristic reasoning about uncertainty. *AI Magazine*, **4**(3):17–23, 1983.

[24] P. Cohen and M. Grinberg. A framework for heuristic reasoning about uncertainty. In: *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI–83)*, pp. 355–357, 1983.

[25] G.F. Cooper and E. Herskovits. A Bayesian method for constructing Bayesian belief networks from databases. In: B. D'Ambrosio, P. Smets and P. Bonissone (eds.), *7th Conference on Uncertainty in Artificial Intelligence*, pp. 86–94. Morgan-Kaufmann, 1991.

[26] G.F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, **9**:309–347, 1992.

[27] A. Darwiche. Dynamic Jointrees. In: *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 97–104. Morgan Kaufmann, 1998.

[28] A.D. Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistical Society Series B*, **41**:1–31, 1979.

[29] A.P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, **2**:25–36, 1992.

[30] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In: *Proceedings of the Twelthth Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pp. 211-219, 1996.

[31] R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[32] J. deKleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, **32**(1):97–130, 1987.

[33] F.J. Díez. Local conditioning in Bayesian networks. *Artificial Intelligence*, **87**:1–20, 1996.

[34] L. David. Genetic algorithms and simulated annealing. London, Pitman, 1987.

[35] M. Dorigo and L. Gambardella. Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, **1**:53–66, 1997.

[36] M. Dorigo and G. Di Caro. The Ant Colony System Meta-Heuristic. In: D. Corne, M. Dorigo and F. Glover(eds.), *New Ideas in optimization*, pp. 11–32, McGraw-Hill, 1999.

[37] J. Doyle. A truth maintenance system. *Artificial Intelligence*, **12**:231-272, 1979.

[38] D.L. Draper. Clustering Without (Thinking About) Triangulation. In: *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pp. 125–133, 1995.

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

[40] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, **4** :507–518, 1975.

[41] M.J. Flores. Incremental compilation of a Bayesian network. Master Thesis, MSc. in Computer Science, Department of Computer Science. Aalborg University, 2000.

[42] M. J. Flores and J. A. Gámez. Triangulation of Bayesian networks by retriangulation. *International Journal of Intelligent Systems* **18**(2):153–164, 2003.

[43] M. J. Flores, J. A. Gámez and K. G. Olesen. Incremental Compilation of Bayesian networks. In: *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI–03)*, Morgan Kaufmann. pp. 233–240, 2003.

[44] M. J. Flores, J. A. Gámez and K. G. Olesen. Incremental Compilation of Bayesian networks in practice. In: *Proceedings of the 4th International Conference on Intelligent Systems Design and Applications (ISDA–04)*, pp. 843–848, Budapest (Hungary), 2004.

[45] M. J. Flores, J. A. Gámez and S. Moral. Abductive inference in Bayesian networks: finding a partition of the explanation space. In: *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU–05)*, Lecture Notes in Artificial Intelligence (LNAI), volume 3571, pp. 63–75, Springer Verlag, 2005.

[46] J.A. Gámez. *Inferencia abductiva en redes causales (Abductive inference in casual networks)*. Doctoral thesis, Dpto. de Ciencias de la Computación e I.A. Universidad de Granada, June 1998.

[47] J.A. Gámez and J.M. Puerta. Searching for the best elimination sequence in Bayesian networks by using ant colony optimization. *Pattern Recognition Letters.*, **23**:261–277, 2002.

[48] J.A. Gámez. Abductive inference in Bayesian networks: A review. In: J.A. Gámez, S. Moral, and A. Salmerón, editors, *Advances in Bayesian Networks*, pp. 101–120. Springer Verlag, 2004.

[49] D. Geiger, A. Paz and J. Pearl. Axioms and algorithms for inferences involving probabilistic independence. *Information and Computation*, **91**:128–141, 1991.

[50] E.S. Gelsema. Abductive reasoning in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters*, **16**:865–871, 1995.

[51] V. Gogate and R. Dechter. A Complete Anytime Algorithm for Treewidth In: *Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence (UAI–04)*, pp. 201–208, AUAI Press, 2004.

[52] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989.

[53] D. Heckerman, D. Geiger and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. In: *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pp. 293–301, 1994.

[54] P. Heggernes and Y. Villanger. Efficient Implementation of a Minimal Triangulation Algorithm. In: *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms.* Lecture Notes in Computer Science, Vol. 2461, pp. 550–561, Springer Verlag, 2002.

[55] M. Henrion and M.J. Druzdzel. Qualitative propagation and scenario-based schemes for explaining probabilistic reasoning. In: P.P. Bonissone, M. Henrion, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pp. 17–32. Elsevier Science, 1991.

[56] L.D. Hernández. Diseño y validación de nuevos algoritmos para el tratamiento. de grafos de dependencias (Validation and design of new algorithms to dependency graph processing.). Doctoral thesis, Dpto. de Ciencias de la Computación e I.A. Universidad de Granada, March 1995.

[57] L.D. Hernández, S. Moral and A. Salmerón A Monte Carlo Algorithm for Probabilistic Propagation in Belief Networks based on Importance Sampling and Stratified Simulation Techniques, *International Journal of Approximate Reasoning* 18: 53–91, 1998.

[58] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.

[59] HUGIN Expert A/S. *API manual for the Hugin Decision Engine V6.3.* http://developer.hugin.com/documentation/API_Manuals/

[60] F.V. Jensen, S.L. Lauritzen and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, **4**:269–282, 1990.

[61] F.V. Jensen and F. Jensen. Optimal junction trees. In: *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pp. 360–366, Morgan-Kaufmann, 1994.

[62] F.V. Jensen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2001.

[63] U. Kjærulff. Triangulation of graphs - algorithms giving small total space. Technical Report R 90-09, Department of Mathematics and Computer Science. Institute of Electronic Systems. Aalborg University, March 1990.

[64] U. Kjærulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, **2**:7–17, 1992.

[65] U. Kjærulff. Aspects of efficiency improvement in Bayesian Networks. PhD thesis, Dept. of Mathematics and Computer Science, Aalborg University, 1993.

[66] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In: *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 302–313, Morgan Kaufmann Publishers, 1997.

[67] C. Lacave. Explicación en redes bayesianas causales. Aplicaciones médicas. (Explanations in causal bayesian networks. Medical applications.) Doctoral Thesis, Dpto. Inteligencia Artificial, UNED. December 2002.

[68] C. Lacave and F.J. Díez. A review of explanation methods for Bayesian networks. *The Knowledge Engineering Review*, 17:107–127, 2002.

[69] C. Lacave and F.J. Díez. Knowledge acquisition in Prostanet, a Bayesian network for diagnosing prostate cancer *Knowledge-Based Intelligent Information and Engineering Systems*, Lecture Notes in Computer Science (LNCS), Volume 2774, pp. 1345–1350, Springer Verlag, 2003.

[70] P. Larrañaga. Aprendizaje estructural y descomposición de redes Bayesianas via algoritmos genéticos (Structural learning and decomposition of Bayesian newtroks by means of genetic algorithms). Doctoral thesis. Dpto. de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco, Donostia, 1995.

[71] P. Larrañaga, C.M. Kuijpers, M. Poza and R.H. Murga. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing*, **7**:19–34, 1997.

[72] K. B. Laskey and S. M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In: *Proceedings of the Thirteenth Conference on*

Uncertainty in Artificial Intelligence, pp. 302–313, Morgan Kaufmann Publishers, 1997.

[73] S.L. Lauritzen, T.P. Speed and K. Vijayan. Decomposable graphs and hypergraphs. *Journal of the Australian Mathematical Society* Series A **36**: 12–29, 1984.

[74] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J.R. Statistics Society. Series B*, **50**(2):157–224, 1988.

[75] S.L. Lauritzen, A.P. Dawid, B.N. Larsen and H.G. Leimer. Independence properties of directed markov fields. *Networks*, **20**:491–505, 1990.

[76] H.G. Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, **113**:99-123, 1993.

[77] V. Lepar and P.P. Shenoy. A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions. In: *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 328–337, 1998.

[78] Z. Li and B. D'Ambrosio. An efficient approach for finding the MPE in belief networks. In: *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pp. 342–349. Morgan Kaufmann, 1993.

[79] Z. Li and B. D'Ambrosio. Efficient inference in Bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, **11**(1):55–81, 1994.

[80] A.L. Madsen and F.V. Jensen. Lazy propagation in junction trees. In: *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 362–369, 1998.

[81] A.L. Madsen and F.V. Jensen. Lazy Propagation: A Junction Tree Inference Algorithm based on Lazy Evaluation. *Artificial Intelligence*, **113** (1-2):203-245, Elsevier Science Publishers, North-Holland ,1999.

[82] S. M. Mahoney and K. B. Laskey. Network engineering for complex belief networks. In: *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pp. 389–396, 1996.

[83] S. M. Mahoney and K. B. Laskey. Constructing Situation-Specific Networks. In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 370-377, 1998.

[84] S. M. Mahoney and K. B. Laskey. Representing and Combining Partially Specified CPTs. In: *Proceedings of the Fifteen Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pp. 391-400, 1999.

[85] J. W. Myers, K. B. Laskey, and K. A. DeJong. Learning bayesian networks from incomplete data using evolutionary algorithms. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pp. 458–465, Morgan Kaufmann 1999.

[86] Z. Michalewicz. *Genectic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[87] R. E. Neapolitan. *Probabilistic Reasoning in Expert Systems. Theory and Algorithms*. Wiley Interscience, New York, 1990.

[88] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[89] http://www.norsys.com

[90] D. Nilsson. An algorithm for finding the most probable configurations of discrete variables that are specified in probabilistic expert systems. MSc.Thesis, University of Copenhagen, Copenhagen, Denmark, 1994.

[91] D. Nilsson. An efficient algorithm for finding the M most probable configurations in Bayesian networks. *Statistics and Computing*, **8**:159–173, 1998.

[92] K. Murphy, Y. Weiss, and M. Jordan. Loopy-belief Propagation for Approximate Inference: An Empirical Study. In: *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 467–475, 1999.

[93] K.G. Olesen and A.L. Madsen. Maximal prime subgraph decomposition of bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics*, Part **B**:(32), 21-31, 2002.

[94] J.D. Park and A. Darwiche. Solving MAP exactly using systematic search. In: *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligene (UAI–03)*, pages 459–468, 2003.

[95] J.D. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, **21**:101–133, 2004.

[96] J. Pearl. and A. Paz. Graphoids: a graph-based logic for reasoning about relevance relations. Technical Report, Cognignitive Science Laboratory, University Of California, Los Angeles, 1985.

[97] J. Pearl. A constraint-propagation approach to probabilistic reasoning. In: L.N. Kanal and J.F. Lemmer (eds.), *Uncertainty in Artificial Intelligence*, pp. 357–370. North Holland, 1986.

[98] J. Pearl. Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, **29**:241–288, 1986.

[99] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, 1988.

[100] Y. Peng and J.A. Reggia. A probabilistic causal model for diagnostic problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, **17**(2):146–162, 1987.

[101] B.W. Peyton. Minimal Orderings Revisited *SIAM Journal on Matrix Analysis and Applications*, **23**(1):271–294, 2001.

[102] A. J. Pfeffer. Probabilistic Reasoning for Complex Systems. PhD Thesis, Stanford University, January 2000.

[103] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, **13**(1-2):81-132, 1980.

[104] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, **32**(1):57–95, 1987.

[105] D. Rose. A graph theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: R. Reed ed. *Graph Theory and Computing*, pp. 183–217, Academic Press, New York, 1972.

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

[107] R.Y. Rubinstein *Simulation and the Monte Carlo Method* John Wiley & Sons, Inc., New York, USA, 1981.

[108] A. Salmerón, A. Cano and S. Moral. Importance sampling in Bayesian networks using probability trees. *Computational Statistics and Data Analysis*, **34**:387–413, 2000.

[109] B. Seroussi and J.L. Goldmard. An algorithm directly finding the k most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning*, **11**:205–233, 1994.

[110] R. D. Shachter,B. D'Ambrosio and B. Del Favero. Symbolic probabilistic inference in belief networks. In: *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 126-131, MIT Press, 1990.

[111] R. D. Shachter, S. K. Andersen and P. Szolovits. Global Conditioning for Probabilistic Inference in Belief Networks In: *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 514-522, Morgan-Kaufmann, 1994.

[112] G.R. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[113] G.R. Shafer and P.P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, **2**:327–352, 1990.

[114] P.P. Shenoy and G.R. Shafer. Axioms for probability and belief-function propagation. In: R.D. Shachter, T.S. Levitt, L.N. Kanal and J.F. Lemmer (eds.), *Uncertainty in Artificial Intelligence, 4.*, pp. 169–198. Elsevier Science Publishers B.V. (North-Holland), 1990.

[115] P. P. Shenoy. Binary join trees. In: *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pp. 492–499, 1996.

[116] P.P. Shenoy. Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of Approximate Reasoning*, **17**(2-3):239–263, 1997.

[117] S.E. Shimony. Explanation, irrelevance and statistical independence. In: *Proc. of the National Conf. in Artificial Intelligence*, pages 482–487, 1991.

[118] S.E. Shimony. The role of relevance in explanation I: Irrelevance as statistical independence. *International Journal of Approximate Reasoning*, 8:281–324, 1993.

[119] S.E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, **68**:399–410, 1994.

[120] K. Shoikhet and D. Geiger. Practical algorithm for finding optimal triangulations. In: *Proceedings of the National Conference on Artificial Intelligence*, AAAI, Menlo Park, CA, USA. pp. 185-190, 1997.

[121] E. Shortliffe. *Computer-Based Medical Consultation: MYCIN*. Elsevier, New York, 1976.

[122] M. Studený. Attemps at axiomatic description of conditional independence. *Kybernetika*, **25**:72–79, 1989.

[123] R.E. Tarjan. Maximum cardinality search and chordal graphs. Unpublished lecture notes, Standford University, 1976.

[124] R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graps, text acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, **13**:566–579, 1984.

[125] R.E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, **55**:221–232, 1985.

[126] T. Verma and J. Pearl. Causal networks: Semantics and expressiveness. In: *Proceedings of the 4th AAAI Workshop on Uncertainty in Artificial Intelligence*, pp. 69–78 Minneapolis, North-Holland, 1988.

[127] Y. Villanger. LEX M versus MCS-M. Technical Report Reports in Informatics 261, University of Bergen, Norway, 2004.

[128] W.X. Wen. Optimal decomposition of belief networks. In: P.P. Bonissone, M. Henrion, L.N. Kanal and J.F. Lemmer (eds.), *Uncertainty in Artificial Intelligence 6*, pp. 209–224. North-Holland, 1991.

[129] Y. Xiang. Multiply sectioned bayesian belief networks for large knowledge systems: an application to neuromuscular diagnosis. PhD Thesis. Department of Computer Science, University of South Carolina, 1992.

[130] Y. Xiang, B. Pant, A. Eisen and M.P. Beddoes. Multiply sectioned bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, **5**:293–314, 1993.

[131] Y. Xiang, D. Poole and M.P. Beddoes. Multiply sectioned bayesian networks and junction forests for large knowledge-based systems *Computational Intelligence*, **9**(2):171–220, 1993.

[132] Y. Xiang and F. Jensen. Inference in multiply sectioned bayesian networks with extended shafer-shenoy and lazy propagation. In: *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pp. 680–687, Morgan Kaufmann Publishers, 1999.

[133] Y. Xiang. *Probabilistic Reasoning in Multiagent Systems. A graphical models approach.* Cambridge University Press, Cambridge, UK, 2002.

[134] Y. Xiang. Comparison of multiagent inference methods in multiply sectioned bayesian networks. *International Journal of Approximate Reasoning*, **33**: 235–254, 2003.

[135] N.L.Zhang and D. Poole Exploiting Causal Independence in Bayesian Network Inference. *Journal of Artificial Intelligence Research*, **5**:301–328, 1996.

Within the field of **Artificial Intelligence** (AI), Expert Systems stand out due to their proven utility and their numerous applications. These systems, which try to *imitate* human experts in a certain knowledge domain, will need to manage the uncertainty inherent in most real life problems. One successful tool to treat uncertainty is the Probability Theory, which gives rise to **Probabilistic Expert Systems** (PES).

Bayesian networks can be located in this PES framework. They provide a quite powerful formalism that gives a representation of the modelled world, which is intuitive (graph structure) and adaptable (belief update). Another appealing feature is their capability of being constructed either by means of experts' contribution or automatically from data, or both.

In a general scheme of an Expert System, the **Bayesian network** (BN) is equivalent to the ***Knowledge Base*** indicating both variable relationships (presence/absence of graph arcs) and their *strength* (probability distributions). BNs answer *queries* also in the form of probabilities: given some observed facts, the user will want to know the resulting posterior probabilities for some other unobserved factors/variables of the problem. That is what basically **inference** in Bayesian networks will attempt to do. Moreover, the search of explanations for those given facts can also be of interest (**abductive** inference).

Different and various inference methods, both approximate and exact, have been proposed in the literature. Nevertheless, those using a secondary structure called junction or join tree are quite broadly applied. The Join Tree (JT) is built from the corresponding BN and can be seen as the ***Inference Engine*** of the expert system. The steps necessary to perform this construction are included in a process called *compilation*. The complexity of compilation increases with the number of variables and depends on the graph structure. Triangulation means a particular compilation stage that is practically unavoidable and presents an NP-hard problem.

Throughout this thesis, three main and distinct inference processes have been analysed in depth and we have proposed **new approaches and algorithms to enhance these procedures:**

- **Triangulation by Re-Triangulation**
- **MPSD-based Incremental Compilation**
- **Plug & Play Object Oriented Bayesian Networks**
- ***Explanation Tree*-based abduction**

S.I.M.D.
Sistemas Inteligentes y Minería de Datos

UCLM

I3A