UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

BAYESIAN REASONING MODULE FOR BDI AGENT ARCHITECTURES. APPLICATION FOR DIAGNOSIS IN FTTH NETWORKS

JESÚS LÓPEZ MÉNDEZ

2011

Proyecto fin de carrera

Título	Bayesian Reasoning Module for BDI agent architectures. Application for diagnosis in FTTH networks
Autor	Jesús López Méndez
Tutor	Álvaro Carrera Barroso
Ponente	Carlos Á. Iglesias Fernández
Departamento	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente	Mercedes Garijo Ayestarán
Vocal	Carlos Á. Iglesias Fernández
Secretario	Ignacio Soto Campos
Suplente	Luis Enrique García Fernández

FECHA DE LECTURA Y DEFENSA

CALIFICACIÓN

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

BAYESIAN REASONING MODULE FOR BDI AGENT ARCHITECTURES. APPLICATION FOR DIAGNOSIS IN FTTH NETWORKS

JESÚS LÓPEZ MÉNDEZ

2011

Resumen

El presente proyecto pretende desarrollar una infraestructura de razonamiento distribuido para sistemas multiagente. En especial, el proyecto se ha centrado en el diagnóstico de fallos en redes de telecomunicación. Dada la incertidumbre durante el diagnóstico y en la distribución de los datos, en este proyecto se ha continuado un proyecto anterior que proponía el uso de redes bayesianas para gestionar dicha incertidumbre.

En este proyecto, abordamos una de sus limitaciones: la incapacidad de gestionar una única red bayesiana en redes de grandes dimensiones. Con este fin, se ha trabajado en el estudio de Redes Bayesianas Multiseccionadas, que facilitan la distribución de una red bayesiana en diversos nodos.

El proyecto ha propuesto un mecanismo de comunicación entre los nodos para mantener su coherencia, que ha sido implementado mediante un middleware de multicasting JGroups. Con el fin de validar este modelo, se ha desarrollado la aplicación al diagnóstico en un escenario FFTH, integrando el sistema de razonamiento en una plataforma multiagente que se despliega sobre un escenario FTTH simulado.

Palabras Clave

MSBN, distribuido, razonamiento, Bayesiano, probabilístico, diagnóstico, MAS, agentes

Abstract

This project aims to provide an infrastructure for distributed reasoning over multiagent systems. In particular, the project has focused on fault diagnosis in telecommunication networks. Given the uncertainty over diagnosis and the distribution of data, this project has continued an earlier project, which proposed the use of Bayesian networks to manage uncertainty.

In this project, we deal with one of its limitations: the inability to manage a single Bayesian network in large communication networks. For this purpose, we have worked on the study of Multiply Sectioned Bayesian Networks, which facilitate their distribution in different nodes.

The project has proposed a communication mechanism between the nodes to maintain consistency, which has been implemented using a JGroups middleware multicasting. To validate this model, it has been applied to diagnosis in FTTH scenario, integrating the reasoning system in a multi-agent platform deployed over a simulated FTTH scenario.

Keywords

MSBN, distributed, reasoning, Bayesian, probabilistic, diagnosis, MAS, agents

"You know, sometimes it is the artist's task to find out
how much music you can still make with what you have left."

Nov. 18, 1998, Itzhak Perlman, the violinist

Acknowledgements

I want to thank everyone around me for the encouragement and strength they have given me to carry out this project. I wish to thank all those with whom I have shared experiences in life.

First and foremost, I owe my deepest gratitude to my fiancée Verónica, who has always been by my side and has supported me in both good and bad moments. She has been the motivation that helped me finish my degree and this project, walking strongly towards our new life together.

I would like to thank my family because they have been there day by day and they have put their faith in me.

To my brother David, because in his company I have learned to think and to take apart devices and, I have dreamed so big things like those that we sometimes achieve.

To my brother Samuel, who has taught me the worthiness of daily effort in everything we do. I wish him the most lasting happiness in his future family.

To my parents José María and Isabel, who have stood a son that hardly goes by home, leaves the bed unmade and talks more with the computer than with them. My father has given me an example of how a person should behave, looking on the bright side of things, and has taught me to be careful in everything I do. My mother has taught me to be able to withstand anything and to know what is important at every moment. They have taught me the trust in God above all things.

To my whole family, Grandparents, aunts, uncles, cousins, nephews and nieces. Especially to Raquel, Alicia, Alberto, Pilar, Modesto, Mª Cruz, Mª Amor, and all those relatives who show that distance is just a physical thing.

To my in-laws, which have shown that you can give without expecting anything in return.

To those loved people who are no longer with us: Marucha, Quico, Inesita, Manuela, Jose... We feel that they accompany us from heaven. To Marucha, because she was much more than a grandmother and marked my life in an impressive way. To Jose Arcones, his life was an example to many of us. His love to everyone left a mark on our hearts that can never be erased.

To my childhood friends: Jesús, Raúl, Antonio, Rubén, Sergio, Víctor, Juan, Pablo, Ana, ... They filled my world and that is why I will never forget them.

To my high school teachers: Juan Manuel, Pepe and Paco. They taught me that lessons are not always written in the books… but almost always.

To my friends from the conservatory: Arturo, Daniel, Manuel, Cristina, María, René, Roberto, Asís, Alba, Sandra, … They helped me discover the world of music and accompanied me during that important time of my life.

To the Franciscan friar Emilio, and Sisters of the Cross, Aurora and Francis and the rest of their communities. They have been my spiritual support and have guided me in the way of faith.

I am indebted to my many of my classmates who support me every day. Especially to Emilio, José Luis, Jaime and Sandra, Juanjo, Valerio, César, Jesús and Elena, Enrique, Laura, Eugenio, Cecilia, José Antonio, Enrique, Diego, Paula, Benjamín, … They have been my family for long periods of time and they have shown me that even if you try to escape, there is always someone waiting for you or looking for you impatiently.

To those friends in higher grades that have helped me with advice and experience to take decisions throughout my degree. Especially to Carlos, Iván and Pablo.

To my friends from weddings, musicals and other special moments: Mónica, Mª Cruz, Rubén, Alicia, Dani, Gema, Ángel, Andrés, Miguel Ángel, Vicente, Carolina, Luismi, … I never truly though such small events could unite people so much.

I am grateful to my Friends: David, Claudia, Inés, Ana, Rafa, Diego, Irene, Iris, Álvaro, Blanca, María, Hellen, Sergio, Alex, … We have lived through countless adventures together, and I hope to live many more to enjoy your great company once and again.

I would like to thank my tutor Álvaro Carrera for all the effort he has devoted to this project. For all hours he has dedicated to the improvements to this system. I wish him a successful future in the new challenges that he will face.

I cannot finish without saying how grateful I am to professors Carlos Á. Iglesias, Mercedes Garijo, Ignacio Soto, Gregorio Fernández, José Ignacio Izpura, Javier Ferreiros. They have shown me that there is still much left for me to learn. Especially to Carlos, because he has shown me that the most important thing is to have faith in yourself and that things are never as tough as they seem.

Finally, I would like to thank all people that I forgot to mention, as well as say that the development of this project has been a pleasure in the company of such a wonderful people.

Contents

Resumen v
Palabras Clave v
Abstract vii
Keywords vii
Acknowledgements xi
Contents xiii
I - Introduction 1
I - 1. Project motivation 2
I - 2. Summary of proposed solution 2
I - 3. Structure of this project report 3
II - State of the art 5
II - 1. Introduction 6
II - 1.1. Distributed Reasoning 6
II - 1.2. Reasoning under Uncertainty 7
II - 1.3. Agent Paradigm and BDI model 7
II - 2. Reasoning Techniques 8
II - 2.1. Case-based reasoning 8
II - 2.2. Rule-based systems 9
II - 2.3. Fuzzy logic 10
II - 2.4. Bayesian Reasoning 11
II - 3. Distributed Reasoning with uncertainty 12
II - 3.1. Multiply Sectioned Bayesian Networks 12
II - 3.2. Distributed Perception Networks 12
II - 3.3. Prior / Likelihood Decomposable Models 15
II - 3.4. Multiply Entity Bayesian Networks 16
II - 4. Bayesian Networks 18
II - 4.1. Definition 18
II - 4.2. Inference in Bayesian Networks 18
II - 4.3. Hugin Architecture: Junction Tree of a Bayesian Network 21
II - 4.4. Building Bayesian Networks 25
II - 4.5. Inference Engines 26

II - 4.6. Directed Cycles in Graphical Models 32
II - 5. Multiply Sectioned Bayesian Networks in detail 34
II - 5.1. MSBN Framework 34
II - 5.2. MSBN Phases 38
II - 5.3. Compilation process 38
II - 5.4. Synchronous architecture for Single-Agent MSBN 41
II - 6. Distributed Communication Frameworks 42
II - 6.1. JGroups 42
II - 6.2. Hazelcast 43
III - Analysis 45
III - 1. Scenario 46
III - 2. Use Cases 46
III - 2.1. Actors 46
III - 2.2. Use Case 1: Loading and operation of the MSBN 47
III - 2.3. Use Case 2: Adaptation of the system 48
III - 3. Requirements 50
III - 3.1. Functional Requirements 50
III - 3.2. Non-Functional Requirements 53
III - 3.3. Requirements Summary 57
III - 4. Tools comparison 58
III - 4.1. Reasoning Techniques Comparison 58
III - 4.2. Distributed Reasoning: MSBN, our choice 64
III - 4.3. BN Inference Frameworks: UnBBayes, our choice 65
III - 4.4. Distributed Communication Frameworks: JGroups, our choice 66
IV - Architecture and Design 67
IV - 1. System parts 68
IV - 2. Developed MSBN Architectures 68
IV - 2.1. Synchronous Architecture for Multi-Agent MSBN 69
IV - 2.2. Iterative Architecture for Multi-Agent MSBN 78
V - Test Plan 95
V - 1. Test Specification 96
V - 1.1. Unit Tests 96
V - 1.2. Integration Tests 98
V - 1.3. Adaptability Tests 102

V - 2. Test Results 102
VI - Case study: FTTH 105
VI - 1. FTTH Scenario 106
VI - 2. Proposed reasoning system 107
VI - 3. Proposed simulation model 108
VI - 4. MSBN for the case study 109
Conclusion 113
Bibliography 115
Glossary 117
Appendix 1. Developer manual 119
1. Subversion 119
2. Maven 120
Appendix 2. Installation manual 121
1. Install Java JDK6 121
Windows: 121
Ubuntu: 121
2. Install Maven 121
Windows: 121
Ubuntu: 121
3. Install Eclipse and proper plugins 121
4. Prepare the project and install dependencies: 122

I - Introduction

In this chapter, a short introduction to this project is given.

First, we show the motivation to the development of this project and why this project is interesting for the field of communications.
Second, a summary of the developed solution is given to ease the reading and understanding work.

Finally, we show a short description of the structure followed in this report, to facilitate the reading of this document.

I - 1. Project motivation

Communication networks are growing more and more in the recently times. Thus, the difficulty of knowing the entire global network state and different states of each of its elements has increased exponentially. Therefore, the diagnosis of a fault in one of these networks is often a process characterized by high complexity and frequently requires the performance of skilled operators. In addition, the fact that both the source and possible solution to a problem detected in the network, a service or a device, a large percentage of cases, is beyond the reach of those who detected their symptoms, the most obvious example of this is an end user who detects a service.

Given the situation discussed in the previous paragraph, it becomes evident the usefulness of a tool for network self-management and service. This tool detects and resolves problems automatically without requiring any user intervention or operator. Still, clearly the complexity of the system will try to simulate an experienced operator is not trivial.

A management tool needs to perform different processes, which could be considered independent, although involving the same aim. In simple terms, these processes are analyzing the state of the network and its elements in each moment, if given any sign of failure or problem, it would happen to the diagnosis and, finally, solving the problem for system, network or device to return to a state of normalcy and proper functionality whenever possible.

The ideas exposed above, the purpose of this final project is the development of a distributed diagnostic system using MSBNs to reach a list of likely causes of failure. This idea can translate this idea into three simple ideas:

- Development of a MSBN based distributed system that allows distributed reasoning
- Exposing how the use of MSBNs allows drawing conclusions at all times during handling the uncertainty inherent in a diagnostic process.
- Development of a distributed system for diagnosis without overloading the network hotspots and thus have greater scope in obtaining the causes of failure in FTTH networks.

Potential applications include decision support to cooperative human users in uncertain domains and troubleshooting a complex system by multiple knowledge based subsystems.

I - 2. Summary of proposed solution

Starting from a centralized MSBN architecture, we develop two architectures that allow distributed reasoning by mean of synchronous and iterative methods respectively. The starting architecture was designed to use shared objects between its nodes. This way, its distribution across a network was not allowed.

We have developed other architectures to enable the compilation and inference by message passing. For this purpose, our system uses an interface that can be easily implemented depending on the communications platform to use.

Inference is performed by mean of two different methods. The first, synchronous method, performs the updating of beliefs on the whole network at the same time. This method has big inconvenient when dealing with scalability and more complex networks. That is the reason why another improved architecture has been developed. The second architecture uses iterative methods that allow a more robust behavior that can adapt to circumstances and events occurred in FTTH networks or other conflictive environments in which this architecture could be used.

Before performing inference, a compilation process is needed to be done. About this process of compilation, several architectural changes are proposed and introduced in the developed architectures.

Thereby, the system developed can be considered to be at the forefront of the current state of the art in the field of distributed Bayesian networks.

I - 3. Structure of this project report

In this section, we will introduce the main points of this report to do easier its reading.

First, we briefly review the reasoning and techniques currently available. Paying more attention to distributed techniques, as well as to the techniques that have been chosen for the development of this project.

Second, the analysis of the problem is done, showing the use cases, the requirements and the reasons why the technologies chosen have been taken.

Third, the developed architectures are described and a detailed view of the whole process is shown.

Forth, we show the test plan and the cases chosen to the verification of the developed architecture.

Fifth, a description of the case study used is given, as well as a short introduction to FTTH Scenario.

Finally, we expose the conclusions that result from the development of this project and the future work that can be done to continue the research in this field.

II - State of the art

In this chapter, the more important available technologies related with this project are presented. Firstly, a summarized introduction to our whole work is shown. Secondly, the more important techniques in reasoning and distributed reasoning can be seen. Thirdly, a more detailed introduction to the techniques used to develop this project is presented. Finally, a short introduction to possible communication frameworks is shown.

In this chapter, all these techniques are not evaluated or compared. The comparison of them and the conclusions extracted from all information given in this chapter are shown in the next chapter.

II - 1. Introduction

Nowadays, many complex environments are part of the business core for a telecommunication operator. This is the case, for example, of network and service management. It involves different processes and they are performed by complex system in complex environments, understanding complex element like an element that:

1. Has many number of parts
2. Properties of these parts are distributed in a heterogeneous way
3. These parts interact through different element in a non-trivial way
4. These parts are adaptable
5. These parts are evolutionary

Many issues are presented when a system has to deal with these features. For this reason, it is needed a distributed reasoning technique that be able to handle uncertainty, to maintain the coherence during the reasoning process between distributed nodes, to be able to self-learning, etc.

In this issue, we study different reasoning techniques that have already been developed. We compare several reasoning techniques that can be used with a distributed approach. Finally, we propose a distributed reasoning system using Multiply Sectioned Bayesian Network, which can be used and integrated in a BDI multi-agent system.

II - 1.1. Distributed Reasoning

It is very important, in distributed systems, achieve a distributed way to reason in complex environments and maintain coherence and consistency in the reasoning.

There is an obvious weakness in distributed systems with central reasoning. Data from an isolated area is lost in the reasoning process if, for example, this area is unable to communicate with the central reasoning node. For example, let be a diagnosis system, there are peripheral sensors and central data processing nodes. If these sensors lose the connection with central nodes, their data could be not used in the reasoning process, thus the route cause of failure may be lost in the diagnosis process.

On the other hand, if the reasoning process is distributed in the whole system, the system can maintain private information about critical points and share only high-level information with other entities. Furthermore, with a distributed reasoning approach, a lot of information can keep in the same node where is generated reducing the overload of the communication network[1]. Using the same example that is shown above, the sensors can perform local reasoning to take decisions about possible actions to be performed (test requesting, reconfiguration actions, etc.) even without external connection. Then, the system would have many distributed nodes that communicate between them with high-level information, i.e. with filtered and processed information about all peripheral elements.

II - 1.2. Reasoning under Uncertainty

The greatest difference between human intelligence and the intelligence of other creatures lies in the fact that the former can, with the help of language, carry on knowledge accumulated over thousands of years. So, the uncertainties of intelligence will definitely be reflected in knowledge[2].

Uncertainty is a fundamental and unavoidable feature of daily life. In order to deal with uncertainty intelligently, we need to be able to represent it and reason about it. Uncertainty is ubiquitous to knowledge fusion. Almost any source of primary data carries some degree of uncertainty.

There are several mechanisms or techniques to deal with uncertainty in intelligent systems. Reasoning with uncertainty is a challenge that is solved using Bayes' theory. The most important techniques about uncertainty treatment are studied in this project and, finally, Bayesian Networks are chosen to carry out this work.

Bayesian probability is a principled formalism for representing uncertainty and drawing inferences in the presence of uncertainty. Bayesian methods have been widely applied in multi-sensor data fusion systems. Bayesian networks are popular models for representing and reasoning about problems involving many related hypotheses.

II - 1.3. Agent Paradigm and BDI model

An *intelligent agent* (IA) is an autonomous entity, which observes and acts upon an environment and directs its activity towards achieving goals.

A *Belief-Desire-Intention* (BDI) *agent* is a concrete type of intelligent agent that follows the BDI model. Superficially, BDI model is characterized by the implementation of an agent's *beliefs*, *desires* and *intentions*. It actually uses these concepts to solve a particular problem in agent programming. In essence, it provides a mechanism for separating the activity of selecting a plan from the execution of currently active plans. Consequently, BDI agents are able to balance the time spent on deliberating about plans (choosing what to do) and executing those plans (doing it). A BDI Agent uses its beliefs to deliberate what plans execute to achieve perform its desires.

Although our developed MSBN framework could be used from any other program, not necessarily based on agents' model, we will focus on agent paradigm to make explanations simpler. This way, each subnetwork, the MSBN part, will belong to the beliefs of an agent. The agent will update this subnetwork with all the information and beliefs it obtain. The public part of this subnetwork, the public beliefs, will be shared with other agents, allowing coordination and cooperation between agents.

Using a distributed approach in a Multi-Agent system (MAS), each agent could represent its knowledge in a Bayesian subnetwork. A model to reason and work with distributed Bayesian networks should have the following basic capabilities as a distributed inference engine does:

1. If an agent is isolated from the rest of the system, it can infer its local knowledge based in its local available information. In other words, independence, local robustness and local intelligence.

2. If none of the connections among the agents is damaged, through communication, the state estimations for all of the agents are consistent. In other words, coordination and system consistency.
3. If some agents are missing or isolated from the rest of the system, they can reorganize themselves. Coordinated inferences can be initiated through the reorganized structures. In other words, self-organization and inference automation.

II - 2. Reasoning Techniques

Many reasoning techniques can be used to process information in an intelligent system. For example, some as rule-based systems, case based reasoning (CBR) systems, fuzzy logic systems or Bayesian reasoning systems. A brief description of some of these techniques is shown below.

II - 2.1. Case-based reasoning

Case-based reasoning (CBR), broadly construed, is the process of solving new problems based on the solutions of similar past problems. This reasoning technique find similar cases in a predetermined past case base according to current inputs or observations. Each case typically contains one scenario of a system. CBR has been formalized for purposes of computer reasoning as a process with the following steps[3], [4]:

- Retrieve the most similar cases, which match the current system observations the most. In this step, if several cases are chosen, logics/rules of combining them are required.
- Use the resulting case to try to solve the current problem.
- If some conflicts appear, revise and adapt the resulting case to a new case according to current system observations.
- Add this new case into the case base as a way of self-extension and learning.

In other words, given a target problem, retrieve cases from memory those are relevant to solving it. Map the solution from the previous case to the target problem. Having mapped the previous solution to the target situation, test the new solution in the system and, if necessary, revise. Then, after the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory. We can see it clearly in the following scheme:

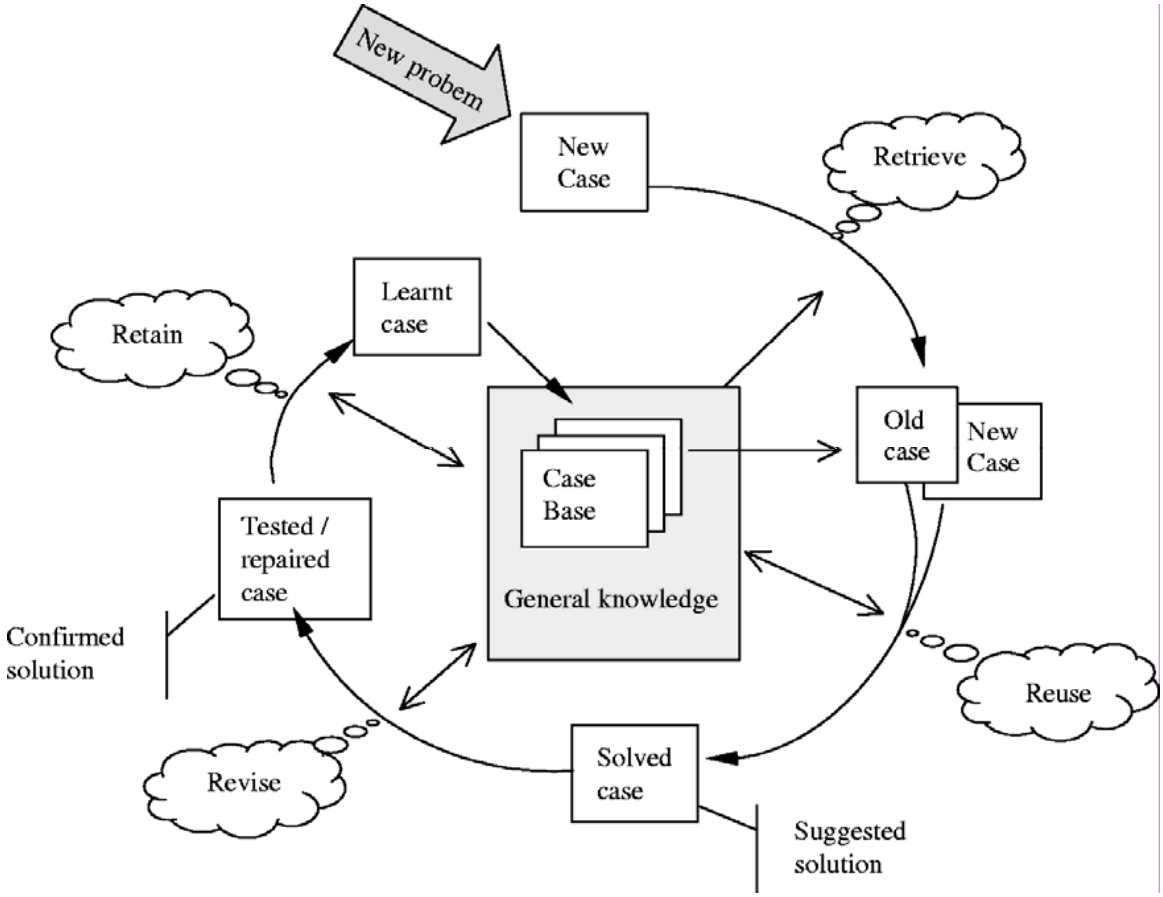


Figure II-1 Case based reasoning steps. Referenced from [4]**.**

CBR works perfectly if the current situation matches one of the stored cases exactly. However, enumerating and storing all possible cases for a complex system is not practical and if the case base were too big, the retrieving process for a similar case would be very slow. Since a case describes one scenario of the whole system, it would not be easy to implement in a distributed way.

II - 2.2. Rule-based systems

In rule-based systems, knowledge is stored as rules in the knowledge base. The inference engine applies the rules to a set of data and obtains conclusions. The knowledge base includes facts and rules representing the knowledge about a particular system from the domain of expertise. A rule indicates a relationship between two facts. Its simplest format is[5]:

IF (Conditions) Then (Facts or Actions)

Two algorithms of inference in rule-based reasoning are often used:

Forward chaining starts with the available data and uses inference rules to extract more data. In other words, forward chaining is a top-down searching procedure. It searches the rule base and when it finds the rule IF part conditions are satisfied, it uses the THEN part as the conclusion.

Backward chaining is the reverse. It is a bottom-up searching procedure. An inference engine using backward chaining would search the inference rules until it finds one that has a consequent (THEN clause) that matches a desired goal.

Searching strategies are very important for a rule based inference engine in terms of efficiency. For a complex system, there are a large number of rules and facts stored in the database to represent possible scenarios for the system. Frequently it results in a slow and computationally expensive reasoning system.

The rule-based engine is a well-developed inference method and there exist many commercial tools, which can be used to develop a rule based inference engine for a particular system. For example, JESS is a rule-based engine-scripting environment written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. JESS uses an enhanced version of the Rete algorithm to process rules. Rete was first designed by Dr. Charles L. Forgy of Carnegie Mellon University in 1974. The Rete algorithm uses a rooted acyclic directed graph to store pattern information. It intends to improve the speed of forward-chained rule systems by limiting the effort required to recompute the conflicted set after a rule is fired.

Rule-based inference engines present some problems when developing a complex system, such as:

- It is almost impossible to list all of the scenarios by using qualitative rules and facts to describe the characteristics of a complex system even when the system is static.
- With a large number of rules and facts, searching the database efficiently is a very challenging task.
- It is very hard to do accurate inference by using a limited set of rules when one part of the system states is not just dependent on particular components under our control.
- It is hard to be distributed for a system with global behaviors. For a loosely coupled complex system, each subsystem can have its own relatively independent rule base and do the inference locally just by exchanging some facts information from other subsystems. However, for a system with global behaviors, by using some local inferences, it is very challenging to keep globally consistent inferences.
- It is difficult to do inference under significant uncertainties by using a pure rule based inference engine.

II - 2.3. Fuzzy logic

Fuzzy logic is a form of many-valued logic derived from fuzzy set theory to deal with reasoning that is robust and approximate rather than brittle and exact. Fuzzy logic variables may have a *truth-value* that ranges in degree between 0 and 1.

An inference engine based on fuzzy logic is used to handle uncertain and imprecise information as an extension of expert inference reasoning method. The basic steps of fuzzy logic reasoning process[6]:

1. Transforms crisp inputs into fuzzy inputs by using corresponding input set membership functions.
2. Search the rule base and fire the matched rules.

3. Combine the matched rules to get one normalized value.
4. Defuzzify the normalized value back to the actual value according to corresponding output set membership functions.

Fuzzy logic allows us to deal with approximate reasoning. However, as an extension of deterministic rule based inference engine, it shares hardly the same problems for reasoning in large-scale complex system as the deterministic rule-based inference engine does. The great difference is that fuzzy logic can use truth-values to introduce beliefs that are not true or false. This is not a solution to deal with uncertainty. Since results are uncertain when inputs are uncertain, the reasoning in complex systems is difficult with fuzzy logic.

II - 2.4. Bayesian Reasoning

Bayesian reasoning systems use Bayesian inference that is a method of statistical inference in which some kind of evidence or observations are used to calculate the probability that a hypothesis may be true, or else to update its previously calculated probability. The term "Bayesian" comes from its use of the Bayes' theorem in the calculation process.

There are several approaches to use Bayesian inference in intelligent systems. Bayesian networks are the key mathematic tool to perform Bayesian inference, but there are two clearly different fields: "Bayesian Networks" and "Distributed Bayesian Networks".

Bayesian Networks

The first one is a centralized approach. A formal definition of a Bayesian network is: "a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG)"[7]. In other words, Bayesian networks are graphical models representing cause-effect relationships among different events. It displays the logic way of how human being thinks.

A graphical model consists of nodes and links. The nodes represent variables and the links between nodes indicate cause-effect relationships. Links have directions, i.e. a link from X to Y is different from a link from Y to X. If a link from X to Y, X is the cause (parent) and$Y$ is the effect (child) and vise versa. Each node can be continuous or discrete with finite number of states. All of the nodes are connected with each other directly or indirectly through inherent relation in the structure of the graph. Such a graphical model can be used to simulate and evaluate how changes in some variables could affect the remaining nodes of the system.

Sometimes, it is difficult to distinguish from which variable is cause and which variable is effect. Under such situations, one can choose either of them subjectively without much effect on the validity of the model.

Bayesian Networks and its related theory will be shown in more detail in section II - 4.

Distributed Bayesian Networks

The distributed approach works as follows [8]. Instead of propagating all of the information everywhere, it is possible to assess first the potential impact of every updating operation on the belief of the target node and to limit the updating process so that only relevant information is propagated. Doing so will decrease the amount of data traffic in the network and the amount of computation expended on interference.

However, it is important that the information we choose not to propagate be allowed to accumulate at the boundaries and discharge its impact to new areas of knowledge once our current set of belief becomes stagnant.

Essentially, a distributed Bayesian network for state inference includes four aspects as a general distributed inference engine does:

- local information processing
- partial intermediate information exchange
- inference global consistency
- self-organization due to partial damage

There are several approaches to Distributed Bayesian Networks, some of which will be shown in the following section II - 3.

II - 3. Distributed Reasoning with uncertainty

Currently, there exist three types of distributed Bayesian networks: Distributed Perception Networks (DPNs), Prior/likelihood Decomposable Models (PLDM) and Multiple Sectioned Bayesian Networks (MSBNs). All of them provide frameworks with different algorithms to partially implement such a conceptual idea of distributed inference engine. In this section, we discuss the three distributed Bayesian networks in detail. In addition, another framework is introduced. Multiply Entity Bayesian Networks (MEBNs) combine the representational power of first-order logic (FOL) and Bayesian Networks (BN).

II - 3.1. Multiply Sectioned Bayesian Networks

A MSBN consists of a set of interrelated Bayesian subnetworks each of which encodes certain knowledge on a subdomain. Bayesian subnetworks are organized into a Hypertree structure such that inference can be performed in a distributed fashion while answers to queries are exact with respect to probability theory.

Each subnetwork only exchanges information with adjacent subnetworks on the Hypertree, and each pair of adjacent subnetworks only exchanges information on a set of shared variables. The great advantage of this organization is complexity of communication among all agents is linear on the number of agents and the complexity of local inference is the same as if the subnet is a single agent based BN.

A more detailed explanation of this distributed reasoning technique is shown in section II - 5 due to MSBN is the reasoning technique chosen for this project.

II - 3.2. Distributed Perception Networks

Distributed Perception Networks(DPNs)are a distributed architecture for efficient and reliable fusion of large quantities of heterogeneous and noisy information [9]. DPNs are composed of numerous agents who cooperate with each other to process systematic reasoning. However, DPNs have a few restrictions, which limit its application region.

DPNs Domain Model

A DPNs domain model M is defined as a tuple $D = (G, V, S, P)$, where G is the set of local DAGs of all DPN agents participating in a particular fusion organization. V is a union of all variables from local clusters and $Q_h \in V$ that contains the hypothesis node. $S = \{S_1, \dots, S_m\}$ is the set of DPN separators and every separator $S_i \in S$ has to be unique and satisfy restriction 1 (see below). P is the set of potentials defined over the DPN domain model and $P_i \in P$ are the potentials for clusterC_i, where $C_i \in T$ and $T = \{T_1, \dots, T_n\}$ satisfying tree architecture and the running intersection property (see II - 4.3).

A local DAG G_i with domain Q_i in a DPN domain model contains a single root nodecorresponding to a service variable $R_i \in Q$ and a set of input variables corresponding to aset of leaf nodes $L_i \in Q_i$. The leaf nodes are always the descendants of the service node.

DPNs have very strict organization constraints, which make its implementation limited to a few situations.

- **Restriction 1**: for any $S_i \in S$, it contains only one unique variable.
- **Restriction 2**: when adding a new agent with its cluster domain C_i, it can connect toonly one unique$C_j \in T$ with a separator S_k, where S_k contains only one variable.
- **Restriction 3**: two local DAGs G_i and G_j with domains Q_i and Q_j respectively canconnect each other if and only if the service variable $R_i \in Q_j$ of G_i is identical to an input variable $R_i \in L_j \subset Q_j$, where L_j is the input variable set of G_j.

If one agent joins the system, it should satisfy these three restrictions at the same time. These three constraints are too strict. For a realistic system, normally, one agent could connect to several agents and an interface (separator set) between two agents contains several variables. A local DAG G_i may contain several root nodes. A node in a separator could be an input variable of one local DAG as well as an input variable of another local DAG.

DPNs deal with three types of agents: static modeling agents, dynamic modeling agents, and appendable modeling agents. Each agent type updates its belief by using a specific algorithm.

Static Modeling Agents

An agent who implements static modeling building blocks can reason in an integrated way about distributions over some quasi-static variables. An event is quasi static if it does not change before the resulting observations are interpreted and used in a decision making process. In another word, a quasi-static event does not involve for a certain period of time after they have been materialized. For example, if a cow were infected, the cow would not be cured during one time step of information fusion process.

Dynamic Modeling Agents

An agent who implements dynamic modeling building blocks can infer from a time series observations. An algorithm for dynamic fusion process is shown in[9]. This algorithm has to satisfy two assumptions:

- All observations are conditionally independent given the sensor propensity.

- The generative model is the same for all observations of a certain type. It means that all of the observations for the same sensor at different time steps are sampled by using the same method and with the same model.

Appendable Modeling Components

An agent implements an additional modeling building block. Such an agent is used to support extensibility for the multi-agent system and makes it flexible and scalable. An algorithm for appendable fusion process is proposed in[9].

All of the agents need to collaborate together to achieve a reasoning task which maps available observations to some hypotheses. To cooperate smoothly and efficiently, two algorithms are used corresponding to two different situations.

Algorithm 1: Top down network configuration

This algorithm is used for self-organization based on the collaboration among multiple agents in a distributed way without any centralized and dominated control agent in order to answer a query of a unique service variable. This algorithm implements a simple self-organization rule: when a query of a service variable is made, it searches the agents containing this service variable. For each of the matched agents, it starts to look for other agents who connect to it by its leaf nodes and satisfy restriction 3. After that, a set of DPNs are formulated, and then use an algorithm similar to collect evidence process introduced in II - 5.4 to calculate the potential of the queried service variable. In the collect evidence process, different agents will use their corresponding algorithms to update their own believes.

Algorithm 2: Bottom-up Network Configuration

This algorithm is suitable for situations where it is desirable to organize fusion systems in response to unusual observations. It implements a simple self-organization rule: when an observation of a leaf variable is available, it searches the agents containing the corresponding service variable of this observed leaf node. For each of the matched agents, it starts to look for other agents who connect to it by its service variable and satisfy restriction 3.

In summary, DPNs as a distributed information fusion system has three limitations discussed in previous context and the following characteristics [9]:

- Reasoning for a single hypothesis variable is processed in a distributed way. The reasoning result reflects the entire available observations and is identical to the result from a single united Bayesian network.
- It does not need to check initial states consistency before the distributed system can work coordinately. Only local models need to be compiled in an independent way prior to run time.
- The information fusion process can work in an asynchronous way, which will improve the speed of reasoning process.
- The information fusion process is automatic and no pre-compilation or on-line check of the formulated structure to guarantee globally consistent inference.

As discussed before, for globally consistent inference in a distributed Bayesian network, there are three prerequisites:

- All Bayesian subnetworks are organized into a tree structure.
- The tree structure satisfies running intersection property.
- Each node shared by two or more sub Bayesian networks should be a d-sepnode.

DPNs satisfies those three conditions implicitly during the formulation of the network due to its strict requirements of individual sub Bayesian network structure and adding new node to the system. More detailed discussion of DPNs can be found in [9].

II - 3.3. Prior / Likelihood Decomposable Models

Prior/likelihood decomposable models (PLDMs) was proposed to infer sensor states and bias from noisy measurable data in large-scale complex distributed sensor networks in [10]. The author pointed out that this method can handle dynamic agent systems, such as adding or deleting agents and several damage situations, e.g., damaged communication links between two agents, bad data caused by failed sensors in a robust way.

In sensor networks, each sensor is an agent. It divides all of the variables in the whole network into two types: observable variables and latent variables. The observable variables are called measurable variables; each measurable variable corresponds to one of the sensors on one of the nodes. The latent variables are actually hidden variables, which are called environment variables. The latent random variables characterize the state of the sensor networks' environment, such as the true temperature, the true pressure, the bias of a sensor itself, etc. All of the measurable variables are children of environment variables and the model needs to specify each measurable variable state probability conditioned on its corresponding hidden variables.

The basic idea of PLDMs is to give each node a subset of local priors. Those subsets of local priors are organized into a junction tree structure (see II - 4.3) called external junction tree structure. In order to increase the robustness of node missing or communication link damages, prior for one node can be distributed to several different nodes as redundancy. The prior of one node is lost only when no nodes that include this node's prior are available. The global prior distribution is obtained through message passing in the external junction tree similar to the normal junction tree belief propagation described in II - 5.4. Message passed between two agents is represented as a Prior/Likelihood factor of the shared variables.

A Prior/Likelihood factor for a set of environment variables C is a pair $(\pi_c, \lambda_c)$where

- $\pi_c$is a prior distribution for C: $prob(C)$.
- $\lambda_c$is a likelihood function: $prob(m|C)$, where m are the observation variables inone node.

In summary, PLDMs has the following limitations for applications other than distributed sensor networks.

- One sensor corresponds to one agent.
- All of the measurable variables are localized.

- All of the variables for the interfaces are belong to the measured variables.
- It doubles quantities of message passing among agents.
- It could break the rules of keeping privacy of each individual agent.
- In order to make globally consistent inferences, pre-compilation or on-line formulation and check of the formulated structure are needed

Detailed discussion of PLDMs can be found in [10].

II - 3.4. Multiply Entity Bayesian Networks

Multi-Entity Bayesian Networks (MEBN) is a first-order probabilistic logic that combines the representational power of first-order logic (FOL) and Bayesian Networks (BN). However, MEBN is still in development, lacking a software tool that implements their underlying concepts.

Ontologies play a major role in semantically aware systems, providing a means for highly effective knowledge sharing. However, they lack a standardized treatment of uncertainty, a ubiquitous feature of multisource fusion problems.

Uncertainty is ubiquitous to knowledge fusion. Almost any source of primary data carries some degree of uncertainty. Bayesian probability is a principled formalism for representing uncertainty and drawing inferences in the presence of uncertainty.

Bayesian networks (BNs) are popular models for representing and reasoning about problems involving many related hypotheses. BNs have been widely applied to information and knowledge fusion, but are fundamentally limited in their expressive power. Specifically, in a standard Bayesian network, all the hypotheses and relationships are fixed in advance, and only the evidence can vary from problem to problem.

Many multi-source fusion problems involve uncertain numbers of interacting entities related to each other in ways that cannot be known in advance. For example, there may be an indeterminate number of weakly discriminatory reports coming from an unknown number of objects, and there may be uncertainty about which report should be associated with which object. This kind of fusion problem produces an exponential set of association hypotheses that require special hypothesis management methods.

MEBN logic combines the flexibility of Bayesian Networks with the representational power of First-OrderLogic [11]. Among other features, MEBN logic can represent and reason with association uncertainty, and thus provides a sound logical foundation for hypothesis management in multi-source fusion.

MEBN represents the world as made up of entities that have attributes and are related to other entities. Knowledge about the attributes of entities and their relation-ships to each other is represented as a collection of MEBN fragments (MFrags) organized into MEBN Theories (MTheories).

MFrag

An MFrag represents a conditional probability distribution of the instances of its resident random variables given the values of instances of their parents in the fragment graphs and given

the context constraints. Random variables are graphically represented in an MFrag either as resident nodes, which have distributions defined in their home fragment, or as input nodes, which have distributions defined elsewhere. Context nodes are the third type of MFrag nodes, and represent conditions assumed for definition of the local distributions.

Typically, MFrags are small, because their main purpose is to model “small pieces” of domain knowledge that can be reused in any context that matches the context nodes. This is a very important feature of the logic for modeling complex, intricate situations and is one that can be seen as the knowledge representation version of the “divide and conquer” paradigm for decision-making. While the latter breaks a hard, complex decision problem in a set of smaller ones, the former uses a similar decomposition approach for representing intricate, complex military situations. This decomposition is accomplished by modeling a complex situation as a collection of small MFrags, each representing some specific element of a simpler situation. The additional advantage of MEBN modeling is the ability to reuse these “small pieces” of knowledge, combining them in many different ways indifferent scenarios.

Indeed, MFrags provide a flexible means to represent knowledge about specific subjects within the domain of discourse, but the true gain in expressive power is revealed when aggregating these “knowledge patterns” to form a coherent model of the domain of discourse that can be instantiated to reason about specific situations and refined through learning. It is important to note that just collecting a set of MFrags that represent specific parts of a domain is not enough to ensure a coherent representation of that domain. For example, it would be easy to specify a set of MFrags with cyclic influences (i.e. a random variable which has its probability distribution influencing itself), or one having multiple conflicting distributions for a random variable in different MFrags (i.e. a random variable with more than one home MFrag, each defining a different distribution).

MTheory

In order to build a coherent model it is important to make sure that a set of MFrags collectively satisfies consistency constraints ensuring the existence of a unique joint probability distribution over instances of the random variables mentioned in the MFrags. Such a coherent collection of MFrags is called an MTheory, and it represents a joint probability distribution for an unbounded, possibly infinite number of instances of its random variables. This joint distribution is specified implicitly through the local and default distributions within each MFrag, together with the conditional independence relationships implied by the fragment graphs.

A generative MTheory summarizes statistical regularities that characterize a domain. These regularities are captured and encoded in a knowledge base using some combination of expert judgment and learning from observation. To apply a generative MTheory to reason about particular scenarios, it is needed to provide the system with specific information about the individual entity instances involved in the scenario. On receipt of this information, Bayesian inference can be used both to answer specific questions of interest and to refine the MTheory. Bayesian inference is used to perform both problem specific inference and learning in a sound, logically coherent manner.

MEBN logic provides a sound mathematical basis for representing and reasoning under uncertainty. PR-OWL uses MEBN's strengths to provide a framework for building probabilistic ontologies, a major step towards semantically aware, probabilistic knowledge fusion systems.

II - 4. Bayesian Networks

Bayesian Networks are computational models that allow reasoning and inference in a similar way as humans do. They are capable of integrate multiple different data sources to achieve a coherent interpretation of these.

II - 4.1. Definition

A Bayesian Network is, in essence, an acyclic directed graph (DAG) which defines a factorization of a joint probability distribution over the variables that are represented by the nodes of the DAG, where the factorization is given by the directed links of the DAG[7].

In other words, a Bayesian Network is a graphical probabilistic model made of **variables** and **cause-effect relations** between them. These variables are the **nodes** of the Bayesian Network and cause-effect relations are represented as directed **edges** linking pairs of nodes. Each variable has a finite set of mutually exclusive **states**.

More formally, let node A be a parent of node B. Using probability calculus, we state the dependence between A and B by means of a **conditional probability table (CPT)** $P(B|A)$. However, if also C is a parent of B, then the two CPTs $P(B|A)$ and $P(B|C)$ alone do not give any clue on how the impacts from A and B interact. That is why we need a specification of a join CPT $P(B \mid A, C)$. This way, to each variable A with parents $B_1, \ldots, B_n$ is attached a conditional probability table $P(A \mid B_1, \ldots, B_n)$[12].

II - 4.2. Inference in Bayesian Networks

Rule-based expert systems have demonstrated their fields of use that was perfectly suited by means of accurate factors or even by manipulating certainty factors. However, they have severe problems when representing incomplete knowledge or reasoning with some degree of uncertainty. When dealing with uncertainty, probability theory is the prevailing method.

Contrary to rule-based systems with certainty factors, inference in Bayesian networks is always consistent. A Bayesian Network (BN) is a knowledge representation scheme as well as provides effective and efficient inference. BNs provide a coherent and effective framework for decision support systems that must function with uncertain knowledge. BNs offer a high level of readability by providing a clear graphical representation while allowing for efficient computations on certain subclasses of networks.

Although inference process in BNs is, a priori, more expensive than in other expert system, efficient inference algorithms have been developed such that inference in Bayesian networks can be done in fractions of a second even for large networks containing hundreds of variables. Efficiency of inference, however, is highly dependent on the structure of the DAG, so networks with a relatively small number of variables sometimes resist exact inference, in which case approximate methods must be applied.

As Bayesian networks most often represent causal statements of the kind $X \rightarrow Y$, where X is a cause of Y and where Y often takes the role of an observable effect of X, which typically cannot be observed itself, we need to derive the posterior probability distribution $P(X \mid Y = y)$ given the observation $Y = y$ using the prior distribution $P(X)$ and the conditional probability distribution $P(Y \mid X)$specified in the model. According to Bayes Theorem for performing this calculation:

$$P(X|Y = y) = \frac{P(y = y|X)P(X)}{P(Y = y)} \qquad \textbf{Eq. II-1}$$

where $P(Y = y) = \sum_x P(Y = y|X = x)P(X = x)$.This rule plays a central role in statistical inference because the probability of a cause can be inferred when its effect has been observed.

More formally, a BN is a triplet (N, E, P). N is a set of nodes. Each node is labeled with a variable associated with a space. We shall use "node" and "variable" interchangeably. Therefore, N represents a problem domain. E is a set of arcs such that $D = (N, E)$ is a directed acyclic graph (DAG). We refer to D as the structure of the BN. The arcs signify directed dependencies between the linked variables. For each node $A_i \in N$, the strengths of the dependencies from its parent nodes π_i are quantified by a conditional probability distribution $p(A_i|\pi_i)$ of A_i conditioned on the values of A_i's parents. For any three sets X, Y and Z of variables, X and Y are said to be *conditionally independent* given Z under probability distribution P if $P(X|YZ) = P(X|Z)$ whenever $P(YZ) > 0$. The basic dependency assumption embedded in BNs is that a variable is conditionally independent of its non-descendants given its parents. This assumption allows P, the joint probability distribution (jpd), to be specified by the product $P = \prod_i p(A_i|\pi_i)$.

Conditional Probabilities

This can be shown in more detail by analyzing the types of rules in Bayesian Networks. Before introducing the most important rule (Chain Rule) in Bayesian networks, we need to introduce evidences/observations and product rule first.

Evidence / observations rule

Evidence / observations are defined as a collection of findings. There are two types of evidences: hard evidence and soft evidence. Hard evidence on variable V is a specification of the value of V, and soft evidence on variable V is a distribution on the values of V. Normally, in most of applications, dealing with hard evidences is enough and most of software can only handle hard evidences.

Product Rule

The Product Rule for probability calculus is the following

$$P(AB) = P(A|\,B)P(B) = P(B|A)P(A) \qquad \textbf{Eq. II-2}$$

Marginalization

Let A be a variable with states $a_1, \dots, a_n$, then $P(A)$ is a probability distribution over these states:

$$P(A) = (x_1, \dots, x_n); \qquad x_i \geq 0; \qquad \sum_{i=1}^{n} x_i = 1, \qquad \textbf{Eq. II-3}$$

where x_i is the probability of A being in state a_i.

From a table $P(A, B)$ of probabilities $P(a_i, b_j)$ the probability distribution $P(A)$ can be calculated. Let a_i be a state of A. There are exactly m different events for which A is in state a_i, namely the mutually exclusive events $(a_i, b_1), \dots, (a_i, b_m)$. Therefore

$$P(a_i) = \sum_{j=1}^{m} P(a_i, b_j) \qquad \textbf{Eq. II-4}$$

This calculation is called **marginalization** and we say that the variable B is marginalized out of $P(A, B)$ (resulting in $P(A)$). The notation is

$$P(A) = \sum_{B} P(A, B) \qquad \textbf{Eq. II-5}$$

Chain Rule

Chain rule is formed by successively applying product rule. It is described in the following. For a Bayesian network, its overall space consists of $U = \{V_1, V_2, \dots, V_n\}$, then

$$P(U) = P(V_1, V_2, \dots, V_n) = \prod_{i=1}^{n} P(V_i | parents\ of\ V_i) \qquad \textbf{Eq. II-6}$$

Let $o_1, o_2, \dots, o_n$ be sets of evidences/observations, then the joint probability including the observations is

$$P(V_1, V_2, \dots, V_n, o_1, o_2, \dots, o_m) = \prod_{i=1}^{n} P(V_i | parents\ of\ V_i) \prod_{i=1}^{m} o_i \qquad \textbf{Eq. II-7}$$

And, normalizing the result we obtain

$$P(V_1, V_2, \dots, V_n \mid o_1, o_2, \dots, o_m) = \frac{P(V_1, V_2, \dots, V_n, o_1, o_2, \dots, o_m)}{\sum_{V_i} \dots \sum_{V_n} P(V_1, V_2, \dots, V_n, o_1, o_2, \dots, o_m)} \qquad \textbf{Eq. II-8}$$

For more details, see [13] page 22.

II - 4.3. Hugin Architecture: Junction Tree of a Bayesian Network

Among all variations developed, HUGIN is the most efficient junction tree-based architecture. The Global Propagation method used in the HUGIN architecture is arguably one of the best methods for probabilistic inference in BNs.

As shown in section II - 4.1, a Bayesian Network (BN) defined over a set of variables is a directed acyclic graph (DAG) augmented with a set of conditional probability distributions (CPDs). More precisely, each variable is represented as a node in the DAG and is associated with a CPD $P(x_i|\pi_i)$, where π_i denotes the parents (also called family) of node x_i in the DAG of the BN. The product of the CPDs in a BN defines the joint probability distributions (JPD) for the BN as:

$$P(U) = p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i|\pi_{x_i}) \qquad \textbf{Eq. II-9}$$

where n the total number of nodes is present in the BN and $p(x_i|\pi_{x_i})$ is the CPD for variable x_i in the BN.

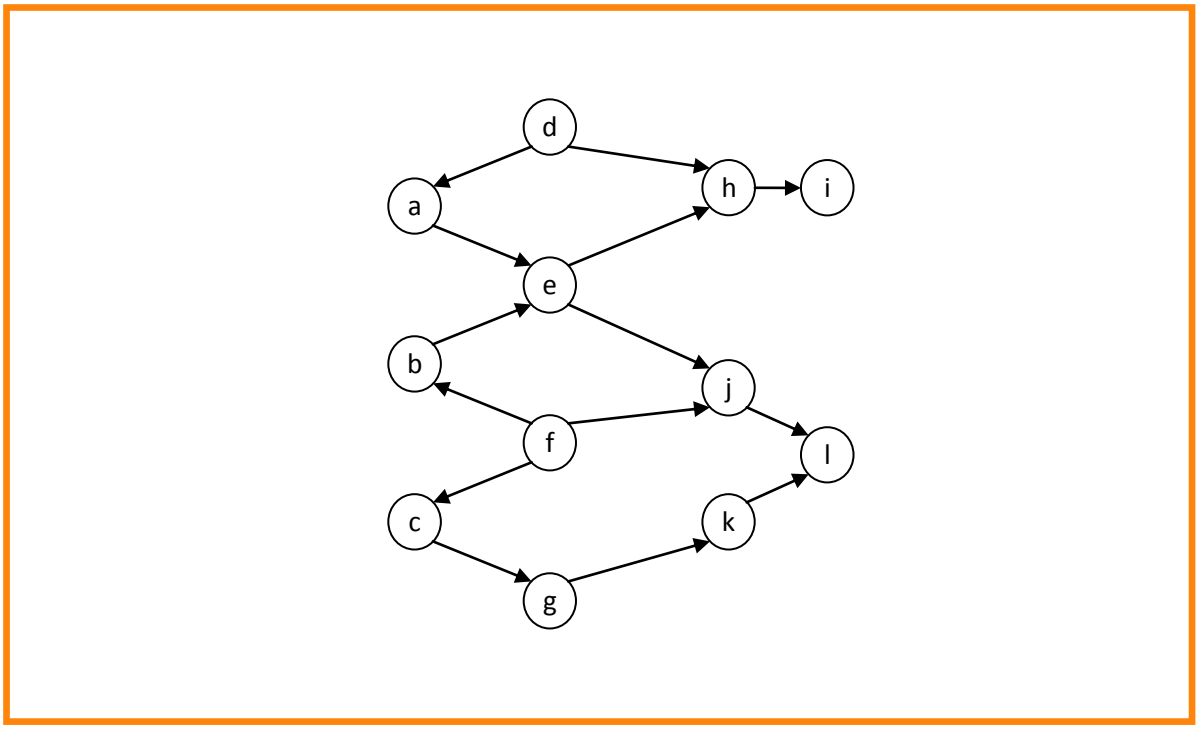


Figure II-2 A notional Bayesian network example

In HUGIN architecture for probabilistic inference, a BN is first transformed into a secondary structure called junction tree.

Junction Tree

A Junction Tree (JT) is an undirected tree constructed from a BN whose nodes are *clusters* (also called *cliques*) of variables (from the original BN). Given two clusters in JT, C_i and C_j, every node on the path between them contains their intersection $(C_i \cap C_j)$. A Separator S_{ij} in JT is associated with each edge and contains the variables in the intersection between neighboring clusters.

Phases

HUGIN architecture consists of several phases, the moralization, the triangulation, the initialization (of the clusters in the JT) phase, and the propagation phase. The Global Propagation method used in the propagation phase for performing message passing is well received and implemented.

Before the propagation, initialization is done to obtain the potential for each cluster in the JT. Then with the Global Propagation method, each cluster potential is transformed into cluster marginal through passing messages with its neighboring clusters.

The efficiency of belief updating in Bayesian networks is very important for probabilistic inferences. Establishing an efficient belief-updating algorithm is fundamental to the application of Bayesian networks. Hugin Algorithm is among the most efficient methods known for belief updating in Bayesian networks in the state of the art. For this, a simple introduction to this algorithm is given below.

Hugin belief updating algorithm[14] includes a few steps: moralization, triangulation, joint tree formulation and full propagation.

Moralization A moral graph of a Bayesian network is an undirected graph, which connects any pairs of variables being members in any $dom(\phi_i)$ existing in the Bayesian network.

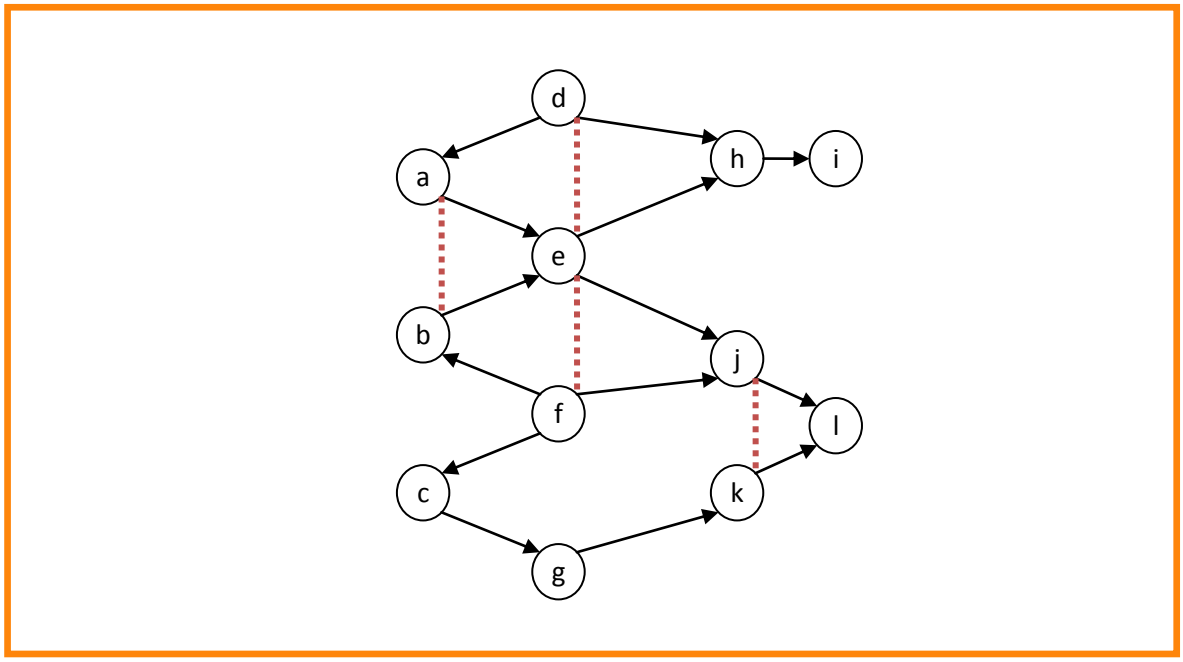


Figure II-3 A otional Bayesian Network after Moralization

Triangulation In order to introduce the idea of triangulation, first, the concept of *perfect elimination sequence* is introduced. When eliminating a node V_i in a Bayesian network, we work with the product of all potential with V_i in the domain. The domain of this product consists of V_i and all of its neighbors in the moral graph. When V_i is eliminated, the resulting potentials has all of V_i's neighbors in its domain and all of the variables in this new domain need to be connected pare wise.

When some nodes are eliminated, new links could be introduced if this node is the link between one or more node, this links are called *fill-ins*. The introduction of *fill-ins* indicates that a potential of a new domain is presented when a variable is eliminated. Eliminating all of the variables in a network one by one forms an elimination sequence and an elimination sequence without introducing *fill-ins* is a *perfect elimination sequence*. There are several concepts that have to be introduced:

Complete Nodes Set a set of nodes is complete if all nodes in this set are pair wise linked.

Clique a complete set is a clique if it is not a subset of another complete set, i.e., the maximal complete set contains a set of specific nodes.

Triangulated Graph an undirected graph with a perfect elimination sequence is called a triangulated graph. The procedure to triangulate a graph is shown below:

1. Eliminate a simplicial node V_i (nodes with a complete neighbor set are called simplicial), then this node with its neighbors denoted as F_{V_i} is a clique candidate.

Figure II-4 A notional Bayesian Network after triangulation

3. Keep the clique candidates that are not subsets of any other clique candidates.
4. The resulting set is the set of cliques.

An undirected graph is triangulated if and only if the cliques of this graph can be organized into a join tree.

Running Intersection Property Let T be a cluster tree over domain$U$. We say T has the running intersection property if whenever there is a variable$V_i \in (C_j \cap C_k)$ and V_i is contained in every cluster in the unique path in T between C_j and C_k, where C_j and C_k are two clusters in T[15].

Join Tree Let G be the set of cliques from an undirected graph, and let the cliques of G be organized into a tree T. If T satisfies the *Running Intersection Property*, then the tree T is a join tree.

Junction Tree Let G be a triangulated Bayesian network with a set of potentials Φ. A junction tree for G is a join tree for G with the following further structure: each potential $\phi \in \Phi$ is attached to a clique who contains $dom(\phi)$, each link has the appropriate separator attached, each separator contains two mailboxes (one for each direction)[16].

There are three key processes for *belief propagation in junction trees*: Collect Evidence, Distribute Evidence and, the combination of both, Full Propagation Evidence[13].

Initialization Phase

After the JT is built, the initialization phase of the HUGIN architecture sets up the initial potentials for the clusters of the JT. A potential is in fact a non-negative function over a set of variables. In particular, the CPD of each node from the original BN is assigned to a cluster that posses the node itself and its parents. Then, within each cluster, these assigned CPDs are multiplied together to form one single potential for the cluster, i.e., the set of CPDs $p(x|\pi_x)$ assigned to a cluster C_i are combined to form the initial cluster potential $\Phi_{C_i} = \prod_{\{x\}\cup\pi_x\subseteq C_i} p(x|\pi_x)$.

Global Propagation

Then the Global Propagation method begins by choosing an arbitrary cluster as a root cluster from which the propagation is initiated. A JT with n clusters will have to perform$2 * (n-1)$ message passes starting from the leaves, dividedinto two phases.

When a cluster receives messages from all its neighbors except that one towards the root, it is allowed to send a message upwards, and so on until the root cluster has received messages from all its neighbors. This is called the *COLLECT-EVIDENCE*.

Now the root cluster sends a message to all its neighbors, and every cluster receiving a message itself, sends another one to all its neighbors except the one from which it received the message, and so on until the leaves are reached. This is called *DISTRIBUTE-EVIDENCE*.

After these two rounds of message passes, the cluster potentials of the JT become cluster marginals.

Absorb mechanism

A single message passing in the Global Propagation method of HUGIN architecture is from one cluster of the JT to another cluster through the separator. It requires a large number of arithmetic operations.

Consider two adjacent clusters C_1 and C_2 with the separator S_{12} in Figure II-5. The single message passing is shown in Figure II-5, where cluster C_1 sends a message through the separator S_{12} to cluster C_2.

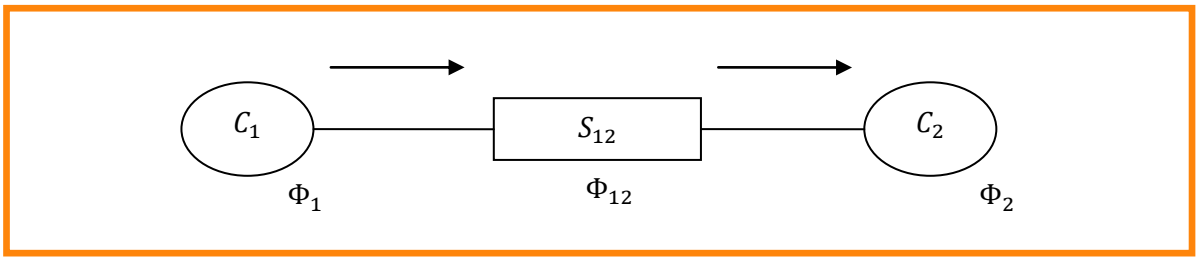


Figure II-5 Single message passing through a separator

Cluster C_1 passing a message to cluster C_2 (or C_2 absorb the message form C_1) means that a two-step computation needs to be done in sequence:

4. Updating the separator potential Φ_{12} by setting

$$\Phi_{12}^{new} = (\sum_{C_1 - S_{12}} \Phi_{C_1}) / \Phi_{12}^{old} \qquad \text{Eq. II-10}$$

5. Updating the cluster potential by setting

$$\Phi_2^{new} = \Phi_2^{old} * \Phi_{12}^{new} \qquad \text{Eq. II-11}$$

The potential new Φ_{12}^{new} is the so-called "message" passed from cluster C_1 to cluster C_2 in the JT. Thus, a single message passed by the Global Propagation method requires three kinds of arithmetic operations: summations, multiplications and a substantial number of divisions. As HUGIN needs $2 * (n-1)$messages to be passed, huge numbers of arithmetic operations are required to process these messages.

As the initialization phase is concerned with the formation of cluster potentials and potentials actively participate in the propagation phase through message passing, the improvement of the initialization phase can also improve the performance of the Global Propagation method of HUGIN architecture.

II - 4.4. Building Bayesian Networks

As described above, a Bayesian network can be described in terms of a qualitative component, consisting of a DAG, and a quantitative component, consisting of a joint probability distribution that factorizes into a set of conditional probability distributions governed by the structure of the DAG. The construction of a Bayesian network thus runs in two phases.

- First, given the problem at hand, one identifies the relevant variables and the (causal) relations among them.
- The resulting DAG specifies a set of dependence and independence assumptions that will be enforced on the joint probability distribution, which is next to be specified in terms of a set of conditional probability distributions, $P(x_i|\pi_{x_i})$, one for each "family", $\{x\} \cup \pi(x)$, of the DAG.

A Bayesian network can be constructed manually, (semi-)automatically from data, or through a combination of a manual and a data driven process, where partial knowledge about structure as well as parameters (i.e., conditional probabilities) blend with statistical information extracted from databases of cases (i.e., previous joint observations of values of the variables). Manual construction of a Bayesian network can be a labor-intensive task, requiring a great deal of skill and creativity as well as close communication with problem domain experts. Extensive guidance on how to manually construct a probabilistic network can be read in [7]. This includes methods and hints on how to elicit the network structure (with emphasis on the importance of maintaining a causal perspective), methods for eliciting and specifying the parameter values of the network, and numerous tricks that can be applied for solving prototypical modeling problems.

Once constructed (be it manually or automatically), the parameters of a Bayesian network may be continuously updated as new information arrives.

Thus, a model for which rough guesses on the parameter values are provided initially will gradually improve itself as it is presented with more and more cases.

II - 4.5. Inference Engines

A huge quantity of Inference engines is available over the world. In this section, we introduce the main Inference Engines for Bayesian Networks with which we have had some experience. Each one has different characteristics and algorithms. We briefly introduce also some of their algorithms and their main features.

II - 4.5.a. SamIam

SamIam is a tool for modeling and reasoning with Bayesian networks developed completely in Java by the Automated Reasoning Group of Professor Adnan Darwiche at UCLA.

SamIam includes two main components: a graphical user interface and a reasoning engine. The graphical interface lets users develop Bayesian network models and save them in a variety of formats. The reasoning engine supports many tasks including: classical inference, parameter estimation, time-space tradeoffs, sensitivity analysis and explanation--generation based on MAP and MPE.

SamIam has a tool called Code Bandit, which is intended to make it easy and fun for Java programmers to learn how to write code based on our inference library Code Bandit writes smart sample code for you, based on settings you configure in SamIam. For example, if you need to use our library to build Bayesian network models, Code Bandit can write a sample program that shows you what methods to call. Code Bandit can also write programs that demonstrate how to execute queries on existing models. Code Bandit is intended to make it easy and fun for Java programmers to learn how to write code based on our inference library.

SamIam supports the Expectation Maximization algorithm for estimating network parameters based on given data. SamIam adopts the "case file" format of Hugin for specifying data as a set of cases. SamIam includes utilities for generating data randomly from a given network and for storing this data in case files.

SamIam supports a number of algorithms for inference in Bayesian networks, such as:

- Three implementations of the join tree algorithm based on: the Hugin architecture, the Shenoy-Shafer architecture, and a new architecture that combines the best of previous architectures.
- An implementation of the Recursive Conditioning algorithm with a time--space tradeoff engine.

SamIam supports non--classical queries, such as the computation of partial derivatives with respect to network parameters. It also supports query--specific inference, which prunes the Bayesian network based on given query before inference algorithms are applied.

SamIam provides an engine for sensitivity analysis in Bayesian networks. The sensitivity analysis engine allows us to specify constraints on network queries and will then identify minimal parameter changes that are necessary to satisfy these constraints. This functionality allows understanding the relationship between local parameters that quantify a Bayesian network, and global conclusions drawn from the network.

SamIam supports opening files in six formats for defining Bayesian networks: the Hugin**.net** format, the Genie **.dsl** and **.xdsl** formats, the Interchange**.dsc** format used by the Microsoft Bayesian Network Toolkit, the Netica**.dne** format and the Ergo **.erg** format.

For Hugin .net files, SamIam supports editing and saving conditional probability table (CPT) values for discrete nodes, the structure of the network, and the appearance of the network.

For each of the other five file types, including Genie .dsl files, SamIam supports editing and saving only the CPT and noisy-or weight definitions. The types of nodes defined within the Genie program that SamIam understands are "chance" nodes defined by CPT or noisy-or semantics and nodes of type "deterministic." SamIam also understands the Genie definitions of nodes as "target," "observation", and "auxiliary", and the definition of states as "target" and/or "default", and Genie sub-models.

II - 4.5.b. Algorithms implemented in SamIam

II - 4.5.b.1. Hugin Algorithm

Hugin Algorithm is among the most efficient methods known for belief updating in Bayesian networks in the state of the art.

SamIam implements a version of Hugin algorithm that uses terminology different from usual. This fact and the fact that SamIam code is not available make we do not give more details about this implementation of the algorithm. However, a more comprehensive description of the usual Hugin Algorithm is shown in section II - 4.3.

II - 4.5.b.2. Shenoy-Shafer Algorithm

Shenoy-Shafer propagation proceeds as follows[17]. First, evidence e is entered into the jointree through evidence indicators. A cluster is then selected as the root and message propagation proceeds in two phases, inward and outward. In the *inward phase*, messages are passed toward the root. In the *outward phase*, messages are passed away from the root.

Cluster i sends a message to cluster j only when it has received messages from all its other neighbors$k$. A message from cluster i to cluster j is a table M_{ij} defined as follows:

$$M_{ij} = \sum_{C_i/S_{ij}} \phi_i \prod_{k \neq j} M_{ki} \qquad \text{Eq. II-12}$$

where Φ_i is the product of CPTs and evidence tables assigned to cluster i.

Once message propagation is finished in the Shenoy–Shafer architecture, we have the following for each cluster i in the jointree:

$$\Pr(C_i, e) = \phi_i \prod_{k} M_{ki} \qquad \text{Eq. II-13}$$

Let us now look at the time and space requirements of the Shenoy–Shafer architecture. The space requirements are simply those needed to store the messages computed by Eq. II-12.

That is, we need two tables for each separator S_{ij}, one table stores the message from cluster i to cluster j, and the other stores the message from j to i. We will assume in our time analysis below the availability of the table ϕ_i, which represents the product of all CPT and evidence tables assigned to cluster i. This is meant to simplify our time analysis, but we stress that one of the attractive aspects of the Shenoy–Shafer architecture is that one can afford to keep this table in factored form, therefore, avoiding the need to allocate space for this table, which may be significant.

As for time requirements, suppose that we have a jointree with n clusters and width w. Suppose further that the table ϕ_i is already available for each cluster i, and let us bound the amount of work performed by the inward and outward passes of the Shenoy–Shafer architecture, i.e., the work needed to evaluate Eq. II-12 and Eq. II-13.

We first note that for each cluster i, Eq. II-12 has to be evaluated n_i times and Equation Eq. II-13 has to be evaluated once, where n_i is the number of neighbors for cluster i. Each evaluation of Eq. II-12 leads to multiplying n_i tables, whose variables are all in cluster C_i. Moreover, each evaluation of Eq. II-13 leads to multiplying $n_i + 1$ tables, whose variables are also all in cluster C_i. The total complexity (since multiplying n elements requires $n - 1$ multiplications) is then:

$$\sum_i O(n_i(n_i - 1)\exp(|C_i|) + n_i \exp(|C_i|)) \qquad \textbf{Eq. II-14}$$

which reduces to $O(\sum_i n_i^2 \exp(w))$, where $\sum_i n_i^2$ is a term that ranges from $O(n)$ to $O(n2)$ depending on the jointree structure.

Given a Bayesian network with n variables, and an elimination order of width w, we can construct a binary jointree for the network with the following properties: the jointree has width $<= w$,and no more than $2n - 1$ clusters.Hence, we can avoid the quadratic complexity suggested above by a careful construction of the jointree, although this can dramatically increase the space requirements.

II - 4.5.b.3. Combination Hugin & Shenoy-Shafer Algorithms

The Hugin and Shenoy–Shafer architectures are two variations on the jointree algorithm, which exhibit different tradeoffs with respect to efficiency and query answering power[17]. The Hugin architecture is more time–efficient on arbitrary jointrees, avoiding some redundant computations performed by the Shenoy–Shafer architecture. This efficiency, however, comes at the price of limiting the number of queries the Hugin architecture is capable of answering. SamIam implements a simple algorithm, which retains the efficiency of the Hugin architecture and enjoys the query answering power of the Shenoy–Shafer architecture.

The combination of the Shenoy–Shafer and Hugin architectures uses zero conscious tables/potentials. The use of these tables provide a simple way to exploit the efficiency of the Hugin method, while extending the set of queries that can be answered efficiently. For the price of a single bit per cluster entry, and some minimal logic operations, all queries answerable using Shenoy–Shafer propagation can now be answered using Hugin type operations.

For applications that require more than just marginal probabilities, such as local search methods for MAP and sensitivity analysis, this can produce a significant speed up over the use of Shenoy–Shafer architecture.

II - 4.5.b.4. Recursive Conditioning Algorithm

Recursive Conditioning, RC, is an any-space algorithm for exact inference in Bayesian networks. It is driven by a structure known as a dtree, which recursively decomposes a network into two smaller subnetworks until the subnetworks only consist of a single CPT. The RC algorithm can then solve each subnetwork independently and merge the localized results together to calculate the desired probability. Many different dtrees exist for a network, and the way the network is decomposed dramatically affects the resource requirements. Therefore, SamIam allows the user to experiment with different dtrees by choosing the elimination order heuristic SamIam uses as the initial step in creating the dtree. More about this algorithm can be read in [18].

II - 4.5.c. UnBBayes

UnBBayes is a probabilistic network framework written completely in Java. It has both a GUI and an API with reference, sampling, learning and evaluation. It supports BN, ID, MSBN, OOBN, HBN, MEBN/PR-OWL, structure, parameter and incremental learning[19].

UnBBayes is an open-source tool, what allows us the implementation of our own code, our own plug-ins or the modification of the existing code. UnBBayes is a plug-in framework that lets you add only that plug-ins that you need for your work.

A plug-in consists of a computer program that interacts with the host application of UnBBayes, to provide specific functionalities. The content of an ordinal UnBBayes plug-in is:

- Plug-in descriptor (XML file)
- Classes (a program)
- Resources (e.g. icons, message files...)

Plug-ins has several benefits, such as reduce the size of each application and organize the system by means of modularization.

II - 4.5.d. Algorithms implemented in UnBBayes

UnBBayes allows the use of several algorithms. For BN, the most important is the Hugin Algorithm, which is described below.

II - 4.5.d.1. Hugin Algorithm

The efficiency of belief updating in Bayesian networks is very important for probabilistic inferences. Establishing an efficient belief-updating algorithm is fundamental to the application of Bayesian networks. Hugin Algorithm is among the most efficient methods known for belief updating in Bayesian networks in the state of the art. For this, a more detailed explanation is given in section II - 4.3.

II - 4.5.e. Genie & Smile

GeNIe is a development environment for building graphical decision-theoretic models. It has been developed at the Decision Systems Laboratory, University of Pittsburgh. They make it available to the community to promote decision-theoretic methods in decision support systems.

GeNIe's name and its uncommon capitalization originate from the name *Graphical Network Interface*, given to the original simple interface to SMILE, our library of functions for graphical probabilistic and decision-theoretic models. GeNIe is an outer shell to SMILE.

GeNIe is implemented in Visual C++ and draws heavily on the MFC (Microsoft Foundation Classes). This makes it not easily portable, although it runs under one of the most popular computing platforms: Windows operating systems. GeNIe allows for building models of any size and complexity, limited only by the capacity of the operating memory of your computer. GeNIe is a developer environment. Models developed using GeNIe can be embedded into any applications and run on any computing platform, using SMILE, which is fully portable.

SMILE (*Structural Modeling, Inference, and Learning Engine*) is a fully platform independent library of functions implementing graphical probabilistic and decision-theoretic models, such as Bayesian networks, influence diagrams, and structural equation models. Its individual functions, defined in SMILE Applications Programmer Interface, allow creating, editing, saving, and loading graphical models, and using them for probabilistic reasoning and decision making under uncertainty.

SMILE is a portable library of C++ classes implementing graphical decision-theoretic methods, such as Bayesian networks and influence diagrams, directly amenable to inclusion in intelligent systems. Its Windows user interface, GeNIe is a versatile and user-friendly development environment for graphical decision-theoretic models.

SMILE also provides Java and .NET wrappers for users who want to use SMILE with languages other than C++. SMILE is equipped with an outer shell, a developer's environment for building graphical decision models, known as GeNIe. GeNIe is platform dependent and runs only on Windows computers. SMILE can be embedded in programs that use graphical probabilistic models as their reasoning engines. Models developed in SMILE can be equipped with a user interface that suits the user of the resulting application most.

Unlike other tools, GeNIe allows associating Properties to networks and nodes. These properties can be used to add information to the Bayesian Network that is not directly related to the probabilistic network but to the real world part modeled in the Bayesian Network.

II - 4.5.f. Netica

Netica is a commercial Bayesian Network tool designed to be simple, reliable, and high performing. Netica is a powerful, easy-to-use, complete program for working with belief networks and influence diagrams. It has an intuitive and smooth user interface for drawing the networks, and the relationships between variables may be entered as individual probabilities, in the form of equations, or learned from data files (which may be in ordinary tab--delimited form and have "missing data"). Values or probabilities may be displayed in a number of different ways, including bar graphs and meters. The knowledge can be transferred between networks by cutting and pasting, or saved in modular form by creating a library of nodes with disconnected links.

Netica can perform various kinds of inference algorithms. Given a new case of which we have limited knowledge, Netica can find the appropriate probabilities for all the unknown variables.

Netica can use influence diagrams to find optimal decisions, which maximize the expected values of specified variables. Netica can construct conditional plans, since decisions in the future can depend on observations yet to be made and the timings and inter-relationships between decisions are considered.

Netica can be used to transform a network in a number of ways. Variables that are no longer of interest may be removed without changing the overall relationships between the remaining variables (technically, the probabilities are "summed out" when we don't know the variable's value, and a more complex operation is used when we do). Probabilistic models may be explored by such operations as reversing individual links of the network, removing or adding causal influences, optimizing one decision at time, etc. These operations may be done with just a click of the mouse, which makes Netica very suitable for easy exploring, and for teaching belief network and influence diagram concepts.

II - 4.5.g. File formats used by inference engines

Each inference engine has been developed to deal with different file formats when saving or loading Bayesian networks. This makes information sharing between them more difficult or use different inference engines for different parts of the same project.

In this point, a recapitulation of those file formats used is done. It is needed to know what inference engines we are going to use before start developing our project because, although conversion between different file formats is not difficult at in general, little information could be lost in the conversion.

The file formats that each inference engine can manage can be read in the following table.

Inference Engine	Data Formats
SamIam	.net (full supported), .dsl, .xdsl, .dsc, .dne, .erg
UnBBayes	.net, .xml (XMLBIF)
Genie & Smile	.xdsl, .dsl, .erg, .dne, .dsc, .net, .dxp
Netica	.dne, .neta, .net, .dxp, .dsc, .ergo

Table II-1 Inference engines data formats

Hugin [.net] Format used by Hugin (Hugin is Copyright © Hugin Expert A/S).

XDSL [.xdsl] This is new XML based native format of the library. For this reason, it is the one that best supports all the features included in SMILE, and is the one we recommend.

DSL [.dsl] This is the Genie &Smile old native format of the library, which is not supported any more. For this reason, all the previous users are strongly advised to switch to new XDSL format.

Ergo [.erg] Format used by Ergo (Ergo is a trademark of Noetic Systems Incorporated). This simple format only supports random variables. You will lose any other information that your network contains when saving using this format.

Netica [.dne] Format used by Netica for net files in text form (Netica is a trademark of Norsys Software Corp).

Neta [.neta] Format used by Netica for net files in binary form (Netica is a trademark of Norsys Software Corp).

Microsoft MSBN [.dsc] Format used by Microsoft Bayes Networks (Microsoft MSBN32 API Library Copyright Microsoft Corp.).

KI [.dxp] Format used by DXpress (DXpress is a trademark of Knowledge Industries).

II - 4.6. Directed Cycles in Graphical Models

As we know, Bayesian networks are directed acyclic graphical models. Bayesian networks cannot handle directed cycles in the model[4]. This makes sense because Bayesian networks are cause-effect models. If there is a directed cycle, the effect becomes the cause and the inference process will be stuck into an infinite loop or some conflicts.

As mentioned before, Bayesian Networks cannot handle cycles, so we need a way to break those cycles but still keep the system characteristics and perform consistent inference reasoning.

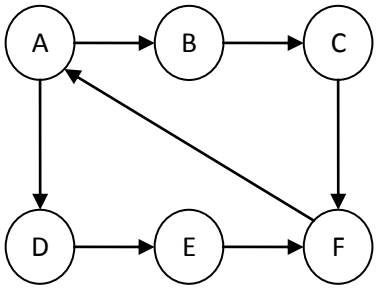


Figure II-6 Example of a Directed Cycle in Bayesian Networks

The first method is to break the directed cycle at one point. For example, if the edge connecting A and F is deleted, the two directed cycles shown in Figure II-6 are broken and the results are shown in Figure II-7. However, by doing that, the information between nodes connected with that edge is lost.

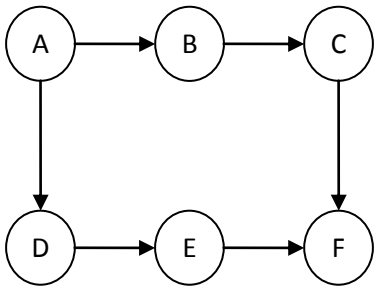


Figure II-7 First Method. Breaking the directed cycle

The second method is to change the direction of one edge in a directed cycle if the directions of some edges are not very important. For example, if the direction of edge between A and F is changed from A to F, there is no directed cycle anymore and the result is shown in Figure #. This method is simple and does not complicate the existed Bayesian network. However, this method is based on the assumption that cause-effect relationship is not significant between two nodes and is limited to certain applications.

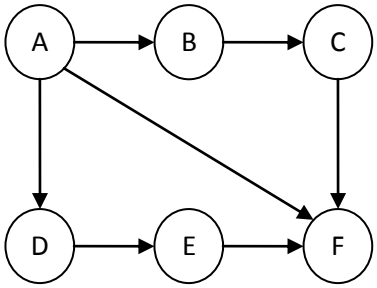


Figure II-8 Second Method. Changing the direction of one edge

The third method is to add an intervention (an instantiation of a node) to break the cycle. As we know, in the sum-product algorithm for discrete Bayesian network inference, when a node V is explicitly instantiated to a specific value, the conditional probability function $P(V|\pi(V))$ will be removed from the sum-product equations. Viewing this removing action graphically, the edges into this specifically instantiated node are deleted from the graphical model. For example, in the simple Bayesian network shown, if node A is instantiated to a specific value, the mass probability function $P(A|F)$ will be removed from the sum-product equation:

$$P(A,B,C,D,E,F) = P(A|F) * P(B|A) * P(D|B) * P(F|D,E) * P(C|A) * P(E|C).$$

Viewing that removal graphically, the directed edge from F to A be removed and the resulted graph is shown in Figure II-9.

Compared with the first method, the resulted graph looks as the same. However, the third method does not lose any information and it requires a node being instantiated. If this node cannot be instantiated, other nodes in the cycle can be candidates to be instantiated and will break the cycle as well. If no node can be instantiated (observed) in a directed cycle, the third method cannot be used. By using the third method, it simplifies the model and keeps the system internal cause-effect relationships.

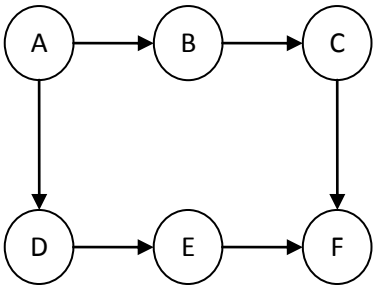


Figure II-9 Third Method: Adding an intervention

II - 5. Multiply Sectioned Bayesian Networks in detail

Bayesian networks (BNs) provide a coherent and effective framework for decision support or diagnosis systems that must function with uncertain knowledge. However, as the problem domains become larger and more complex, modeling a domain as a single BN and conducting inference in it becomes increasingly more difficult and expensive.

Multiply Sectioned Bayesian Networks (MSBNs) provide one alternative to meet this challenge by relaxing the single BN paradigm. The framework allows a large domain to be modeled modularly and the inference to be performed distributively, while maintaining the coherence. The framework can be applied under the single agent paradigm as well as the multi-agent paradigm. It supports hierarchical model based diagnosis and modeling large systems with the object-oriented paradigm.

A MSBN is a set of subnetworks that form a concrete tree structure and share information sharing some nodes between them. Each one of these subnetworks can be contained in any program. In our work, we will associate the concept of program with the *Intelligent Agent* concept. In artificial intelligence, an *intelligent agent* (IA) is an autonomous entity, which observes and acts upon an environment and directs its activity towards achieving goals.

Although our developed MSBN framework could be used from any other program, we will focus on agent paradigm to make explanations simpler.

II - 5.1. MSBN Framework

A BN is a triplet (U, G, P), where U is a set of domain variables, G is a directed acyclic graph (DAG) whose nodes are labeled by elements of U, and P is a joint probability distribution (JPD) over U. G encodes conditional independencies among variables in U.

A form of representation for a BN is Junction Tree (JT) (see II - 4.3). In a JT, each cluster consists of a subset of the domain variables. Each cluster acts as a unit / object in message passing during inference. Similarly, a MSBN partitions a large domain into a **Hypertree** of some natural subdomains. Such subdomains become the units for distribution. A Hypertree can be probed to be a JT.

MSBN partitions a large domain into a Hypertree, which is analogous to a JT of a single BN. This is the first level of application of the JT representation in MSBNs. On the other hand, a cluster in a JT has no internal structure. The belief over a cluster is represented as a **potential** (non-normalized probability distribution) over all variables in the cluster. Since a subdomain in a large domains itself large in general, representing it as a cluster is neither feasible nor necessary. Instead, a MSBN represents each subdomain as a Bayesian network called a subnetwork.

A MSBN M is a collection of Bayesian subnetworks that together defines a BN. M represents probabilistic dependence of a total universe partitioned into multiple subdomains each of which is represented by a subnetwork.

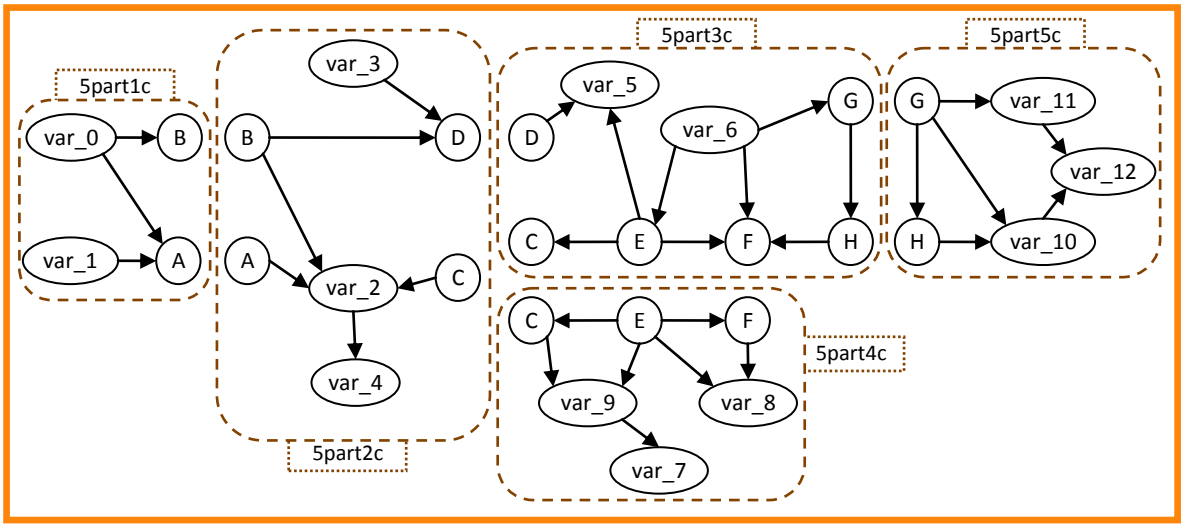


Figure II-10 A notional MSBN called 5partc

In an MSBN M, a set of $n > 1$*Bayesian subnetworks* $U_0, U_1, \dots, U_{n-1}$ populates a total universe U of variables. Each subnet has knowledge over a subdomain $U_i \subset U$ encoded as a *Bayesian subnetworks* (U_i, G_i, P_i). The collection $\{G_0, G_1, \dots, G_{n-1}\}$ of local DAGs encodes subnetwork's knowledge of domain dependencies. Local DAGs should overlap and be organized into a *Hypertree*. Adjacent subnetworks exchange information over their overlapped variables.

The partition should satisfy certain conditions to permit coherent distributed inference. One condition requires that nodes shared by two subnets form a **d-sepset**, as defined below.

> Let $G_i = (N_i, E_i)\ (i = 0, 1)$ be two graphs. The graph $G = (N_0 \cup N_1, E_0 \cup E_1)$ is referred to as the union of $G_0$and G_1, denoted by $G = G_0 \cup G_1$.

In a JT of a single BN, a message sent by a cluster C to an adjacent cluster C' is a belief table over their intersection $C \cap C'$, called *sepset* (which labels the link between the clusters). Like a cluster in a JT, a sepset has no internal structure. In a large domain, the intersection of two subdomains, called a d-sepset, is also large in general. Hence, more compact representation of the d-sepset is desired. The MSBN framework represents each **d-sepset** also as a JT, called a **linkage tree**, which allows a more efficient representation of the message passed between subdomains. This is the third level of application of the JT representation in MSBNs.

D-Sepset

A d-sepset is the set of nodes that are shared between two subnetworks whose parents are in one of the two subnetworks. A more formal definition can be read above.

Let $D_i = (N_i, E_i)\ (i = 0, 1)$ be two DAGs such that $D = D_0 \cup D_1$ is a DAG. Theintersection $I = N_0 \cap N_1$ is a *d-sepset* between D_0 and D_1 if for every node $x \in I$ with its parents π in D, either $\pi \subseteq N_0$ or $\pi \subseteq N_1$. Each node$x \in I$ is called a d-sepnode.

In Figure II-11 above, a example of d-Sepset is shown.

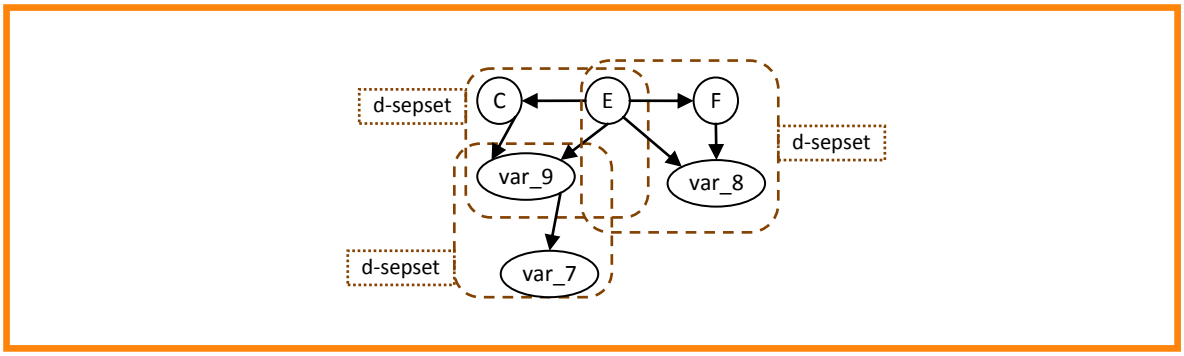


Figure II-11 D-Sepset examples

A d-sepset is a sufficient information channel for passing all relevant evidence from one subnet to another. Formally, a pair of subnets is conditionally independent given their d-sepset.

Hypertree MSDAG

Just as the structure of a BN is a DAG, the structure of a MSBN is a **multiply sectioned DAG(MSDAG)** with a *Hypertree* organization. That is to say, subnetworks in MSBN M are organized in a tree structure, where each subnetwork U_i is known as a **Hypernode** and each existing interface between two H*ypernodes* is known as a **Hyperlink** or **Linkage**. In this graph, nodes will be *Hypernodes*, and edges will be *hyperlinks*. Graphically, a Hyperlink separates the Hypertree MSDAG into two subtrees. Semantically, this corresponds to the conditional independence given the d-sepset.

As it is seen, we can use subnetwork or Hypernode interchangeably, as well as **Linkage** instead Hyperlink.

A Hypertree MSDAG $D = \cup_i D_i$, where each D_i is a DAG, is a connected DAG constructible by the procedure described below.

Building a Hypertree MSDAG

Start with an empty graph (no node). Recursively add a DAG D_k, called a *Hypernode*, to the existing MSDAG$\cup_{i=1}^{k}$ subject to the constraints:

- [d-sepset] For each $D_j (j < k)$, $I_{jk} = N_j \cap N_k$ is a *d-sepset* when only D_j and D_k are considered.
- [local covering] There exists $D_i (i < k)$ such that, for each $D_j (j < k;\ j \neq i)$, we have$I_{jk} \subseteq N_i$.

For an arbitrarily chosen such D_i, I_{ik} is the **Hyperlink** between D_i and D_k which are said to be **adjacent**.

It can be proven that if each Hypernode D_k of a Hypertree MSDAG is replaced by the clusterN_k and each Hyperlink between D_j and D_k is replaced by the d-sepsetI_{jk}, then the resultant isa JT.

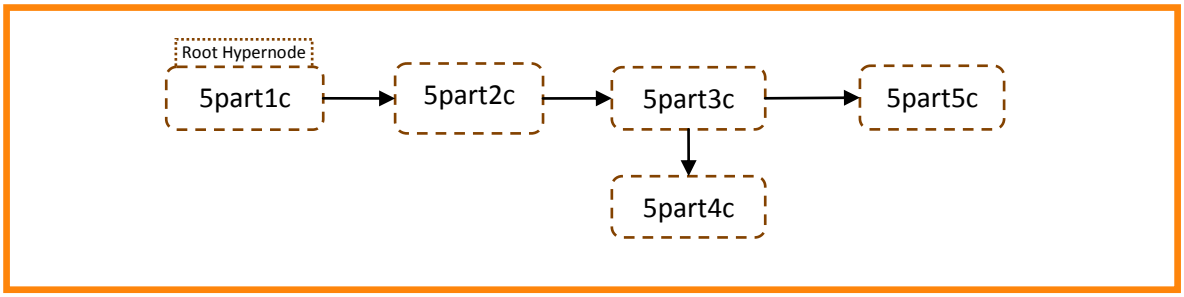


Figure II-12 Hypertree MSDAG for the notional MSBN

Public Nodes

Nodes shared between two Hypernodes are known as public nodes or shared nodes. A more formal definition is shown below.

Let N_i be a Hypernode in MSBN M, and let N_i have a subset of nodes $\{P_i\}$, which are also part of other Hypernodes. Then, each node contained in this subset of shared nodes $\{P_i\}$ is known as a public node.

Hypernode

A Hypernode or Subnetwork is a Bayesian Network. The only particularity of this Bayesian Network is that it is related to other Hypernode by mean of Hyperlink. A Hypernode is part of the Hypertree of the MSBN and it has necessarily to be connected to at least other Hypernode through a Hyperlink.

Link

A link is an association between two cliques from different Hypernodes, which contain the same public nodes. These public nodes shared between both cliques need to form a d-sepset.

A link has three cliques that identify it: *host0*, *clique* and *host1*. Cliques *host0* and *host1* are cliques in the subnetworks associated by this link. Clique *clique* is the d-sepset build from the intersection of *host0* and *host1*.

Let N_k be a Hypernode descendant from Hypernode N_j in the Hypertree. Let C_j be a clique in N_j, and C_k be a clique in N_k. Let I_{jk} be a d-sepset between N_j and N_k, whose nodes are all contained in both $C_j$and C_k. Then, there exists a Link L_{jk_i} that has I_{jk} as *clique* of the link and C_j and C_k as cliques *host1* and *host2* respectively, according to the Hypertree offspring.

Hyperlink

A Hyperlink or Linkage establishes a link between two Hypernodes. Every Hypernode in the Hypertree (in the MSBN) is connected directly or indirectly to any other Hypernode through one or more than one Hyperlink respectively. Each Hyperlink serves as the information channel between subnetworks connected. It is referred to as a subnetwork interface.

Let N_k be a Hypernode direct descendant (child) from Hypernode N_j in the Hypertree. Then, there exists a Hyperlink H, which has all existing links $\{L_{jk_i}\}$ between N_j and N_k in its link list. The parent Hypernode N_j is called the *net0* of this Hyperlink H and child Hypernode N_k is called the *net1* of the Hyperlink H.

Linkage Tree

All d-sepsets in the links of a Hyperlink are associated forming a junction tree known as Linkage Tree.

Let $\{I_{jk_i}\}$ be the set of d-sepset cliques belonging to all existing links $\{L_{jk_i}\}$ in Hyperlink H. Since those links $\{L_{jk_i}\}$ belong to Hyperlink H, set of d-sepset cliques $\{I_{jk_i}\}$ has a Junction Tree structure.

To summarize, the process necessary to allow inference through a MSBN and communication of beliefs between its Hypernodes, require a fairly complex structure. This structure divides the MSBN in Hypernodes or subnetwork, which are connected through Hyperlinks or Linkages. Linkages are made of the public nodes of the subnetworks, and are divided in Links according to the structure of cliques present in each Hypernode.

II - 5.2. MSBN Phases

The process of use of a MSBN is divided in three well-differentiated phases:

1. Subnetworks Load Phase
2. Compilation Phase
3. Query phase

Load phase (Phase 1) is a trivial one and will be shown in short. Nevertheless, Compilation Phase (Phase 2) is the focus of the following section II - 5.3 in which it is explained in more detail.

Load phase involves reading a file with the information about the corresponding Bayesian network/s. With this information, a new object representing a Bayesian Network will be constructed. The type of file and the way of loading the subnetwork depend on the program used as we have described in section II - 4.5.

The number of files to read depends on the architecture used. In single-agent architecture (II - 5.4) all subnetworks are loaded at the same time while, in multi-agent architectures (IV - 2.1, IV - 2.2) each agent load only the subnetwork/s that it own.

II - 5.3. Compilation process

The compilation of the MSBN allows carrying out inference and queries. Similarly as in a simple BN, some processes are needed to be done before perform inference. Those processes are, in summary, the following:

1. Build Hypertree
2. Moralize
3. Triangulate
4. Compile Junction Tree
5. Make Linkage Tree
6. Initialize Beliefs

These processes can vary a little depending on the distributed architecture. However, the role they have is the same for all architectures. Now a more detailed explanation of these processes is given. Only the theory is shown in this section. For a more programmatic detailed explanation, it can be seen sections II - 5.4, IV - 2.1 and IV - 2.2, according to each architecture.

II - 5.3.a. Build Hypertree

To build the Hypertree of the MSBN is assigning the parents and adjacent subnetworks to each subnetwork resulting in a Hypertree structure (see II - 5.1). It is needed to be done in a centralized way. A subnetwork is arbitrary selected as root Hypernode. At the beginning, only the root subnetwork belongs to the Hypertree. Then, recursively, linkages are assigned between the subnetwork that is not in Hypertree yet and has the largest amount of nodes in common with another subnet that is already in Hypertree.

Then, the following loop is executed:

Code II-1 Hypertree Building Loop

```
while (Not all subnetworks in Hypertree) do
   • Calculate the greatest possible d-sepset between a
     subnetwork that is not in Hypertree yet and another that
     belongs to Hypertree.
   • Create a new Linkage between subnetworks chosed.
   • Set the parents and adjacent subnetworks of both
     subnetworks respectively.
end while
```

II - 5.3.b. Moralize

A *V* structure is formed when a node has two parent nodes. An example of V structure is shown in subnetwork 5part2c in Figure II-13. Thus, due to V structure, if evidences of node *var_2 are* fixed, probabilities of *A*'s states will change if the probabilities of *B*'s states change. A fill-in is an undirected arc that indicates dependence between two nodes. In Figure II-13, two fill-ins can be seen between *var_0* and *var_1*, and between *A* and *B,* in subnetworks 5part1c and 5part2c respectively.

The first action a subnetwork needs to do to allow moralization is local moralization. That is, look for all *V* structures and add corresponding fill-ins.

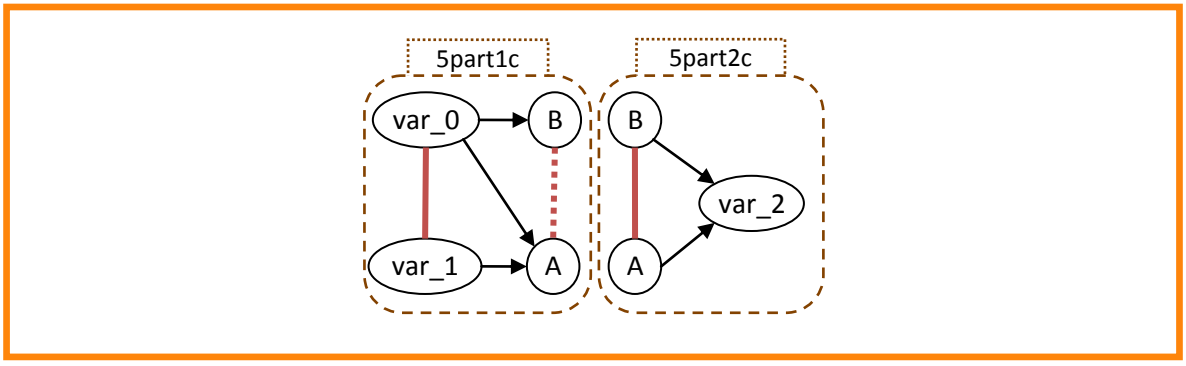


Figure II-13 Moralization Example. V structure.

Then, each fill-in is sent to each subnetwork that has the two nodes linked by the fill-in shared. A subnetwork that receives a fill-in should add this to its M*arkov arcs list*. The M*arkov arcs list* is a list with all arcs added during moralization and triangulation pointing dependencies non-explicitly shown in the original BN. In the example shown in Figure II-13, subnetwork 5part2c need to send a fill-in between $A - B$ to subnetwork 5part1c.

II - 5.3.c. Triangulate

Triangulation, as described in II - 4.3, is a process where an elimination order for the nodes of a subnetwork is defined. The only condition to allow the elimination of a node is that all its adjacent nodes are adjacent to each other. If two nodes adjacent to a node are not adjacent to each other, a new fill-in needs to be added to the *Markov Arcs List* (see II - 5.3.b) of the subnetwork. If the nodes linked by the fill-in are shared with another subnetwork, this fill-in needs to be sent to the other subnetwork to be added.

For example, as we see in Figure II-14, if C, an adjacent of B, is not adjacent of D, another adjacent of B, then a new fill-in between $C - D$ needs to be added in subnetwork 5part2c. This new fill-in needs to be sent and added to subnetwork 5part3c since D and C are shared nodes.

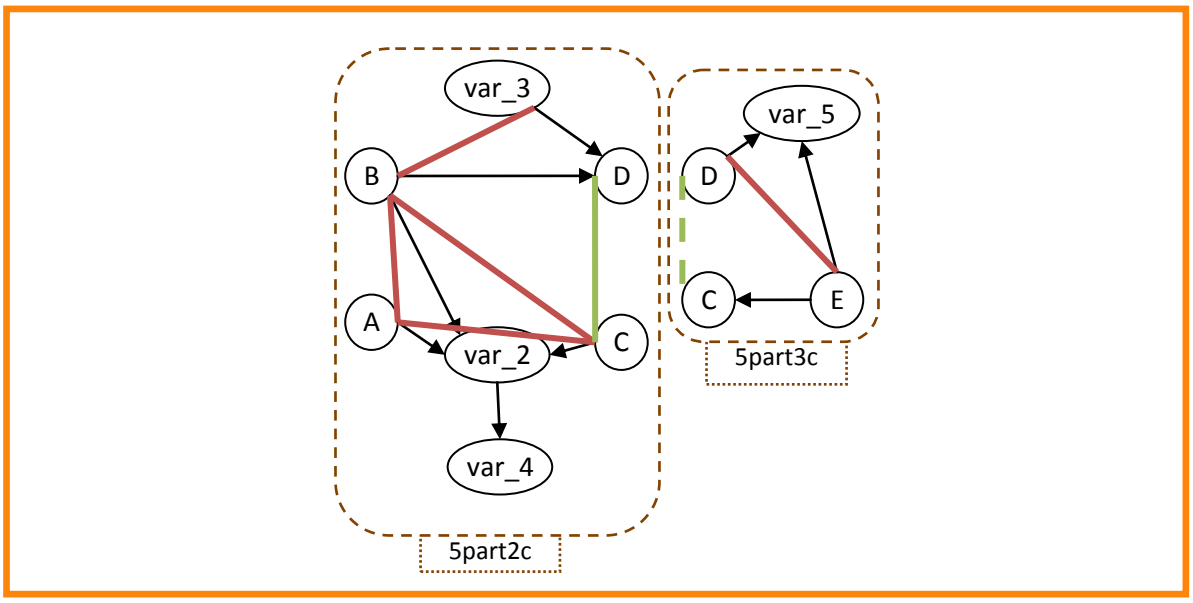


Figure II-14 Triangulation Example. Shown moralization fill-ins in red and triangulation fill-in in green.

II - 5.3.d. Compile Junction Tree

To compile the Junction Tree, a subnetwork needs to associate a separator or clique to each node and initialize tables of separators and cliques as well as marginal probabilities in each node. This is a very condensed description that can be extended with details in sections II - 5.4, IV - 2.1 and IV - 2.2.

II - 5.3.e. Make Linkage Tree

By Making the Linkage Tree of a MSBN, we mean make the Linkage Tree of each Linkage in the MSBN. This process involves the creation of a new JT assigned to each Linkage. In each Linkage, a

Link is created between each corresponding cliques. To each link, a *host0*, *clique* and *host1* cliques are assigned as defined in section II - 5.1. Then, tables in the cliques in the Linkage are initialized. For more details about this process, see sections II - 5.4, IV - 2.1 and IV - 2.2.

II - 5.3.f. Initialize Beliefs

Before performing inference in a MSBN, an initialization of beliefs in all subnetworks is needed. Each subnetwork, as BN that it is, has initial beliefs that do not need to be the same as initial beliefs of another neighboring network with shared nodes. To maintain coherence in the MSBN, a common initialization of common beliefs is needed, which can be achieved by many different algorithms.

The algorithm of initialization used in our work is the same used in UnBBayes MSBN Framework (see II - 5.4). It is that, as is done in Bayesian networks, we need a two-phase initialization: COLLECT-EVIDENCE and DISTRIBUTE-EVIDENCE. A simple representation of this process for a three-subnetwork MSBN is shown in the collaboration diagram of Figure II-15. The subnetwork *subnetwork_1* has the role of root subnetwork in this example.

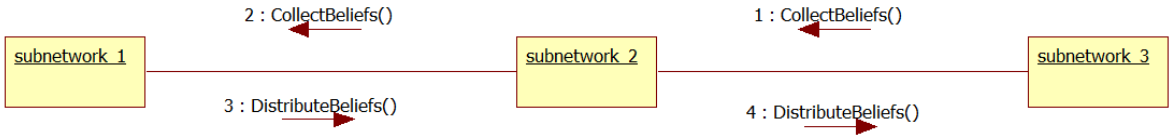


Figure II-15 Belief Initialiazation algorithm. Collaboration Diagram

In MSBN Framework, we obtain different results from initialization depending on which subnetwork initiate the loop (the root subnetwork). This is a convention arbitrary chosen. Other conventions or techniques of initialization can be chosen, but they are beyond the scope of this project.

II - 5.4. Synchronous architecture for Single-Agent MSBN

Probabilistic reasoning in BNs, as commonly applied, assumes a single-agent paradigm. That is, a single processor accesses a single global network representation, updates the joint probability distribution over the network variables as evidence becomes available and answers queries.

This architecture is designed to work in a single machine. All of the updates are made directly, by changing the properties in the corresponding object. In fact, several subnetworks share the same nodes, which are the same object instances.

In this architecture, the greatest advantage in comparison with a simple BN is the modularity offer by the ability of subnetworks to be interchanged or replaced. The modularity improves inference efficiency in a single user oriented system in a large problem domain.

Off-line time

The communication of beliefs in this architecture is achieved in a synchronous way by means of recursive calling to the corresponding methods. That means each subnetwork is waiting while

this updating loop is performed. More about distribution of beliefs can be read in section II - 5.3.f.

The solution proposed to avoid this behavior can be seen in section IV - 2.2.

II - 6. Distributed Communication Frameworks

This project bases its communications in a communication interface, which can be implemented by multiple types of communication frameworks. Thus, the results and architectures designed are independent form the communication framework used.

In fact, both distributed architectures designed have different communication requirements.

- The first architecture, Synchronous one, needs a communication framework that supports synchronous operations. That is, a framework that supports Remote Procedure Call (RPC). It has been manually implemented by the following means:
    - Direct method calling to emulate communication in an only computer.
    - Socket based communication that allows the calling of methods remotely according to the message passed. In addition, some objects can be serialized to be sent.
    - Jadex Agent Platform, which implements itself serializing methods for messages.
- The second architecture, Iterative one, needs a communication framework that supports asynchronous message sending. This means that, when a message arrives, a simple operation (normally queue the information conveniently) needs to be done. It has been achieved using an already developed communication framework:
    - JGroups, which is a reliable group communication toolkit written entirely in Java. It is based on IP multicast.

  Although developed architecture is independent from communication framework, there are several services provided by JGroups toolkit that are not easily found in other communication frameworks, such as:
        - Notification about joined / left /crashed members
        - Point-to-multipoint and Point-to-point messaging
        - State transmission

II - 6.1. JGroups

JGroups is a toolkit for reliable multicast communication. It can be used to create groups of processes whose members can send messages to each other. The main features include

- Group creation and deletion. Group members can be spread across LANs or WANs
- Joining and leaving of groups
- Membership detection and notification about joined/left/crashed members
- Detection and removal of crashed members

- Sending and receiving of member-to-group messages (point-to-multipoint)
- Sending and receiving of member-to-member messages (point-to-point)

Flexible Protocol Stack and Reliable Communications

The most powerful feature of JGroups is its flexible protocol stack, which allows developers to adapt it to exactly match their application requirements and network characteristics. Besides unicast communications, JGroups extends reliable unicast message transmission (like in TCP) to multicast settings. As described above it provides reliability and group membership on top of IP Multicast. Since every application has different reliability needs, JGroups provides a flexible protocol stack architecture that allows users to put together custom-tailored stacks, ranging from unreliable but fast to highly reliable but slower stacks.

Over implemented protocols, reliable communications are achieved including

- lossless transmission of a message to all recipients (with retransmission of missing messages)
- fragmentation of large messages into smaller ones and reassembly at the receiver's side
- ordering of messages, e.g. messages m1 and m2 sent by P will be received by all receivers in the same order, and not as m2, m1 (FIFO order)
- atomicity: a message will be received by all receivers, or none.

II - 6.2. Hazelcast

Hazelcast is an open source clustering and highly scalable data distribution platform for Java, which is:

- Lightning-fast; thousands of operations/sec.
- Fail-safe; no losing data after crashes.
- Dynamically scales as new servers added.
- Super-easy to use; include a single jar.

Hazelcast allows you to easily share and partition your application data across your cluster. Hazelcast is a peer-to-peer solution (there is no master node, every node is a peer) so there is no single point of failure.

Hazelcast is simple. JVMs that are running Hazelcast will dynamically cluster. Although by default Hazelcast will use multicast for discovery, it can also be configured to only use TCP/IP for environments where multicast is not available or preferred. Communication among cluster members is always TCP/IP with Java NIO beauty. The only we need to use Hazelcast is just adding the hazelcast.jar into our classpath and start coding.

III - Analysis

This chapter discusses the analysis process that has been followed to obtain the system requirements. This will show, firstly, the objectives and scope of the project to properly frame the work area.

After presenting the scope of the project, this section shows the use cases and requirements arising from these.

Finally, a brief comparison of the tools presented in section II - is done, showing the reasons why the tools we use in this project are chosen.

III - 1. Scenario

The solution resulting from this project can be used in numerous scenarios to be able to deal with uncertainty in a distributed way. That is the reason why, in this section we will present a general system that can operate reasoning independently of the scenario in which it is situated.

For a more specific description in a concrete scenario, the case study presented in section VI - can be seen.

III - 2. Use Cases

This section identifies general use cases of normal use in the system, in order to get a full specification of expected usage of the system with the aim of establish a complete list of requirements.

Below, the actors identified within the use cases are presented. Later, by mean of use case descriptions and UML diagrams of them, we establish the relationships between the actors and the system.

III - 2.1. Actors

Here we present the actors identified within the use cases that are shown below.

Actor Identifier	Name	Description
ADM	Administrator	It is responsible for starting, loading and shutdown of each node of this reasoning system. Since this is a distributed system, there could be several ADM, each of which is responsible of a node of this system.
USR	User	It is responsible for managing and operating of the system. It can introduce new information about the state of the system or the results of other sources of knowledge, listen to changes in the knowledge of this reasoning system to be able to produce some reactions to these events or make queries or request any information about the collected knowledge at any time, independently from belief updates.

Table III-1 Actors

III - 2.2. Use Case 1: Loading and operation of the MSBN

III - 2.2.a. Description

The ADM responsible for the new Hypernode load it from the information received or loaded from a file. Then, the USR takes the control of this Hypernode and perform compilation and initialization of the Hypernode. The USR is the responsible for the answering of queries about the beliefs and the state of the compilation of the Hypernode, as well as carrying out the updating of beliefs or sending them to allow the updating of beliefs of other Hypernodes. At any moment, ADM can decide to shutdown this Hypernode.

III - 2.2.b. Use Case Specification

UC-1	Loading and operation of the MSBN
Description	The system performs the loading of the MSBN and each Hypernode operates communicating with others and answering to queries.
Actors	ADM, USR
Normal Sequence	Loading of Hypernode Compilation of the Hypernode according to the MSBN Initialization of the beliefs of the Hypernode according to adjacent Hypernodes in the MSBN Receive petitions of belief updating Update beliefs Receive queries about beliefs Receive queries about compilation state Shutdown this Hypernode
Exceptions	Steps from 4 to 7 can be done in different order or even omitted according to USR needs.

Table III-2 UC-1 Specification

III - 2.2.c. Use Case Diagram

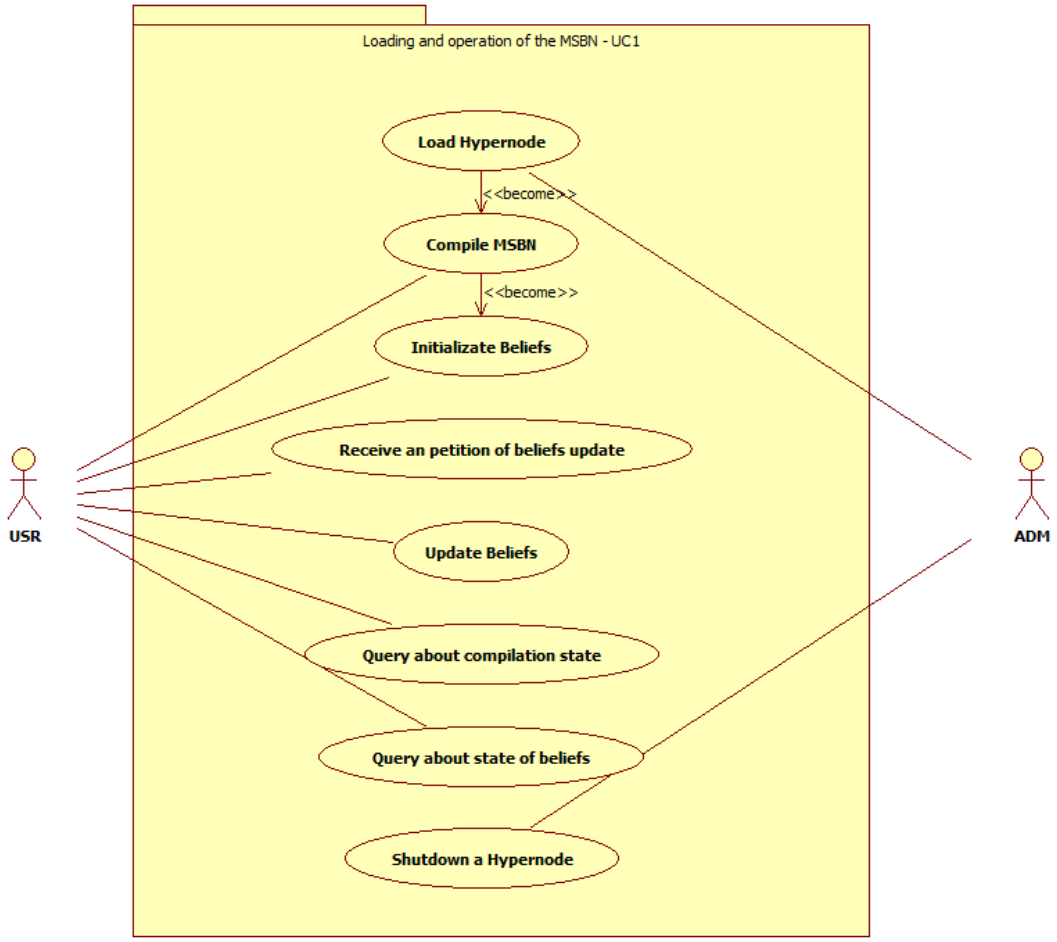


Figure III-1 Use Case Diagram for UC-1

III - 2.3. Use Case 2: Adaptation of the system

III - 2.3.a. Description

An ADM can decide load a new Hypernode or shutdown an existing one at any time, as we saw in UC-1. Thus, the ADMs that are responsible for all other Hypernodes have to adapt the configuration of the MSBN to changes occurred. That is, to adapt the MSBN to a fallen Hypernode, to adapt the MSBN to an appeared Hypernode or to adapt two previously existing MSBN to be joined into an only MSBN. Only in the case that a Hypernode is isolated, the URS is the responsible for the adaptation of this Hypernode to the situation.

III - 2.3.b. Use Case Specification

UC-1	Adaptation of the system
Description	During the whole time the system is running, many events can happen such as the appearing or disappearing of a Hypernode. Since the structure of the MSBN can be affected, adaptation needs to be done by each Hypernode. Each ADM is responsible for the adaptation to the falling or the appearing of a Hypernode, as well as to the joint of two MSBNs into an only MSBN if it becomes possible. On the other hand, USR is responsible for the adaptation of the Hypernode to an isolated situation.
Actors	ADM, USR
Normal Sequence	Adaptation to a fallen node Adaptation to an appeared node Adaptation to an isolated node Adaptation to the joint of two MSBNs into an only MSBN
Exceptions	The order is not relevant in this use case.

Table III-3 UC-2 Specification

III - 2.3.c. Use Case Diagram

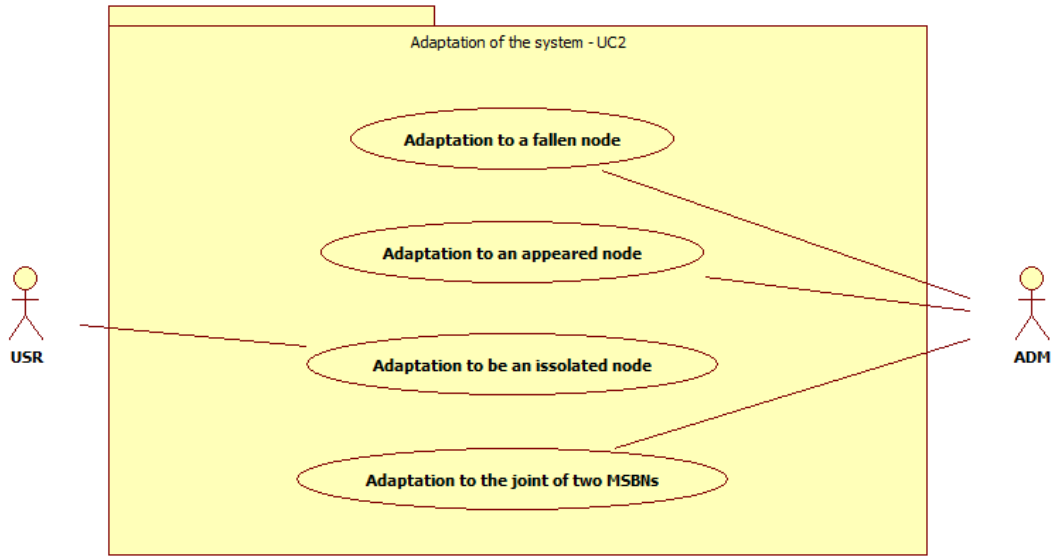


Figure III-2 Use Case Diagram for UC-2

III - 3. Requirements

This section presents the requirements of the system obtained through the use cases presented and the global scenario.

Each requirement is presented in a table, which contains:

- Identifier. An unique Identifier for the requirement. It is expressed in the format:
  - FR-n: Functional requirement number 'n'.
  - NFR-n: Non-functional requirement number 'n'.
- Title. A short name for the requirement.
- Description. A detailed specification of the features that the system should satisfy.
- Priority. Degree of need for meet this requirement. Its value can be one of the following:
  - Essential. It is mandatory for the correct functioning of the system.
  - High. It could be satisfied partially, but it is highly recommended.
  - Medium. Optional requirement, which could improve the quality of the system.
  - Low. Minor requirement which, if met, would add little improvements to the system that may be undetectable by the user.
- Related use cases. Specifies those use cases that, after an detailed analysis, have result on this requirement.
- Related requirements. Shows those requirements with which this requirement is related.

III - 3.1. Functional Requirements

FR-1	Load a MSBN Hypernode
Description	The system should be capable of perform the loading of all Hypernodes of the MSBN. It can be achieved from different ways, such as load from files, load from received messages or load from the ontology of an agent.
Priority	Essential
Related use cases	UC-1
Related requirements	FR-4, FR-6

Table III-4 Load a MSBN Hypernode

FR-2	Compile a MSBN Hypernode
Description	Each Hypernode should be capable of perform compilation.
Priority	Essential
Related use cases	UC-1
Related requirements	FR-5, FR-12

Table III-5 Compile a MSBN Hypernode

FR-3	Sending information
Description	Each Hypernode should be capable of send all information required for other Hypernodes to compile. It should be capable of select only those destination Hypernode that need the information.
Priority	Essential
Related use cases	UC-1
Related requirements	FR-4, NFR-1, NFR-3

Table III-6 Sending information

FR-4	Receiving messages
Description	Each Hypernode should be capable of use properly information received from other Hypernodes.
Priority	Essential
Related use cases	UC-1
Related requirements	FR-3, NFR-3

Table III-7 Receiving messages

FR-5	Recompilation of a Hypernode
Description	After the reception of several types of messages, each Hypernode should be capable of return to a lower step of the compilation. This should be done without losing data or affecting the normal operation of the system.
Priority	High
Related use cases	UC-1
Related requirements	FR-2, FR-8, FR-9, FR-10, FR-11

Table III-8 Recompilation of a Hypernode

FR-6	Stopping a Hypernode
Description	Each Hypernode should be capable of stop its operation. This action can be launched by the same Hypernode or by an external agent.
Priority	Medium
Related use cases	UC-1
Related requirements	FR-1

Table III-9 Stopping a Hypernode

FR-7	Verifying the non-existence of cycles
Description	The system should be capable of verify if there are any cycles in the global BN, throwing an exception in that case. In addition, it should inform of the correction of the graph before compilation finishes.
Priority	Low
Related use cases	UC-1, UC-2
Related requirements	

Table III-10 Verifying the non-existence of cycles

FR-8	Adaptation to an isolated Hypernode
Description	An isolated Hypernode should be capable of reson using all information it has.
Priority	Low
Related use cases	UC-2
Related requirements	FR-5, NFR-10

Table III-11 Adaptation to an isolated Hypernode

FR-9	Adaptation to an appeared Hypernode
Description	After the apparition of a new Hypernode, all Hypernodes in the MSBN should adapt their structures to the new situation.
Priority	Essential
Related use cases	UC-2
Related requirements	FR-2, NFR-10

Table III-12 Adaptation to an appeared Hypernode

FR-10	Adaptation to a fallen Hypernode
Description	Afther the fallen of an already existing Hypernode, all Hypernodes in the MSBN should adapt their structures to the new situation.
Priority	High
Related use cases	UC-2
Related requirements	FR-5, NFR-10

Table III-13 Adaptation to a fallen Hypernode

FR-11	Adaptation to the join of two MSBNs
Description	When the MSBN has been split into several MSBN, after the apartion of the joining Hypernode, the MSBNs should be capable of merge into one again.
Priority	Medium
Related use cases	UC-2
Related requirements	FR-5, FR-12, NFR-10

Table III-14 Adaptation to the join of two MSBNs

FR-12	Split an MSBN
Description	When some existing Hypernodes in an MSBN cannot be connected, the system should be capable of operate as if there were several separated MSBNs.
Priority	Medium
Related use cases	UC-2
Related requirements	FR-5, FR-11, NFR-10

Table III-15 Split an MSBN

III - 3.2. Non-Functional Requirements

NFR-1	Coherence and consistency
Description	The system should be capable of maintain coherence and consistency during the whole reasoning process. In the case of Iterative architecture, the requirement of concistency is temporary lower.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	FR-3

Table III-16 Coherence and consistency

NFR-2	Handle uncertainty
Description	The system should be capable of handle uncertainty inherent in all complex environments.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	

Table III-17 Handle uncertainty

NFR-3	Tolerant to communication failures
Description	The system should be capable of be tolerant to communication failures. This can be done by the using of a framework that provide this feature.
Priority	High
Related use cases	UC-1, UC-2
Related requirements	FR-3, FR-4, NFR-9

Table III-18 Tolerant to communication failures

NFR-4	Incomplete data reasoning
Description	The system should be capable of offering a coherent result even with incomplete data set.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	

Table III-19 Incomplete data reasoning

NFR-5	Scalability
Description	The system should be scalable enough to be useful in a real scenario with a high quantity of nodes.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	NFR-6

Table III-20 Scalability

NFR-6	Privacy
Description	The system should be capable of keep private data in Hypernodes sharing only public information. This is also important for the scalability of the system.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	NFR-5

Table III-21 Privacy

NFR-7	Autonomy
Description	The system should be autonomous. Each Hypernode should be an independent unit capable of reason itself.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	NFR-8

Table III-22 Autonomy

NFR-8	Distributed and decentralized
Description	The system should have a set of Hypernodes that are all equal. Operations are distributed across different parts of the systems.
Priority	High
Related use cases	UC-1, UC-2
Related requirements	NFR-7

Table III-23 Distributed and decentralized

NFR-9	Asynchronous communications
Description	The system should have asynchronous communications. A message should be attended at any time.
Priority	High
Related use cases	UC-1, UC-2
Related requirements	NFR-3

Table III-24 Asynchronous communications

NFR-10	Adaptability
Description	The system should adapt to each scenario and situation. Hypernodes should be capable of adapt the structure of the MSBN (Hypertree) to each situation.
Priority	High
Related use cases	UC-1, UC-2
Related requirements	FR-8, FR-9, FR-10, FR-11, FR-12, NFR-11

Table III-25 Adaptability

NFR-11	Stability
Description	The system should be stable enough to be useful when reasoning in a real scenario.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	NFR-10

Table III-26 Stability

NFR-12	Portability
Description	The system should be portable to allow its use in all those devices that form a FTTH network.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	

Table III-27 Portability

NFR-13	Maintainability
Description	The system should be maintainable to allow future work and modifications over the developed framework.
Priority	Essential
Related use cases	UC-1, UC-2
Related requirements	

Table III-28 Maintainability

III - 3.3. Requirements Summary

In this section, a summarizing table (Table III-29) with requirements and priorities is shown.

Id	Title	Priority
FR-1	Load a MSBN Hypernode	Essential
FR-2	Compile a MSBN Hypernode	Essential
FR-3	Sending information	Essential
FR-4	Receiving messages	Essential
FR-5	Recompilation of a Hypernode	High
FR-6	Stopping a Hypernode	Medium
FR-7	Verifying the non-existence of cycles	Low
FR-8	Adaptation to an isolated Hypernode	Low
FR-9	Adaptation to an appeared Hypernode	Essential
FR-10	Adaptation to a fallen Hypernode	High
FR-11	Adaptation to the joing of two MSBNs	Medium
FR-12	Split an MSBN	Medium
NFR-1	Coherence and consistency	Essential
NFR-2	Handle uncertainty	Essential
NFR-3	Tolerant to communication failures	High
NFR-4	Incomplete data reasoning	Essential
NFR-5	Scalability	Essential
NFR-6	Privacy	Essential
NFR-7	Autonomy	Essential
NFR-8	Distributed and decentralized	High
NFR-9	Asynchronous communications	High
NFR-10	Adaptability	High
NFR-11	Scalability	Essential
NFR-12	Portability	Essential
NFR-13	Maintainability	Essential

Table III-29 Requirements summary

III - 4. Tools comparison

III - 4.1. Reasoning Techniques Comparison

III - 4.1.a. Comparison of reasoning techniques

In this section, we analyze data shown in II - 2. As we have seen, each technique presents its strengths and its weaknesses. To solve the problem stated in section VI - and satisfy requirements showed in section III - 3, we sum up the requirements as follows. We want to find one reasoning technique that allows us to reason with the following requirements:

1. To maintain coherence and consistency in the reasoning using a distributed approach.
2. To handle uncertainty inherent in all complex environments.
3. To be tolerant with communications failures.
4. To keep private data in local nodes sharing only public information.
5. To offer a coherent result even with uncompleted data set.

In Table III-30, a comparison between most relevant reasoning techniques with a distributed approach about this features is shown[4]. As result of this comparative table, we can deduce that Bayesian inference is the technique that meets better the requirements listed above.

Reasoning Technique	Rule systems	CBR	FuzzyLogic	Bayesian inference
Coherence/Consistency	Good	Good	Bad	Good
Handle uncertainty	Null	Null	Good	Good
Failures tolerance	Medium	Bad	Medium	Medium
Maintain private data	Good	Medium	Good	Medium
Uncompleted data set	Bad	Good	Medium	Good

Table III-30 Reasoning techniques comparison

As we see in Table III-30 the best choice is Bayesian Inference. Bayesian reasoning allows handling uncertainty while maintaining coherence and consistency.

Bayesian reasoning is not itself a failure tolerant technique. Thus, architecture used should be responsible for failure handling and recovering techniques.

Bayesian reasoning can maintain data as private if we implement an architecture that allows it. It is not a characteristic of the system, but can easily be achieved. Reasoning when the data set is incomplete is allowed by several techniques.

As we see, Bayesian reasoning is the technique that had better satisfy requirements stated. In consequence, we base this whole project on this reasoning technique.

Another option would be Fuzzy Logic, due to handling uncertainty is our main requirement. In the following section, we carry out a comparison between both techniques to clarify why this decision has been made.

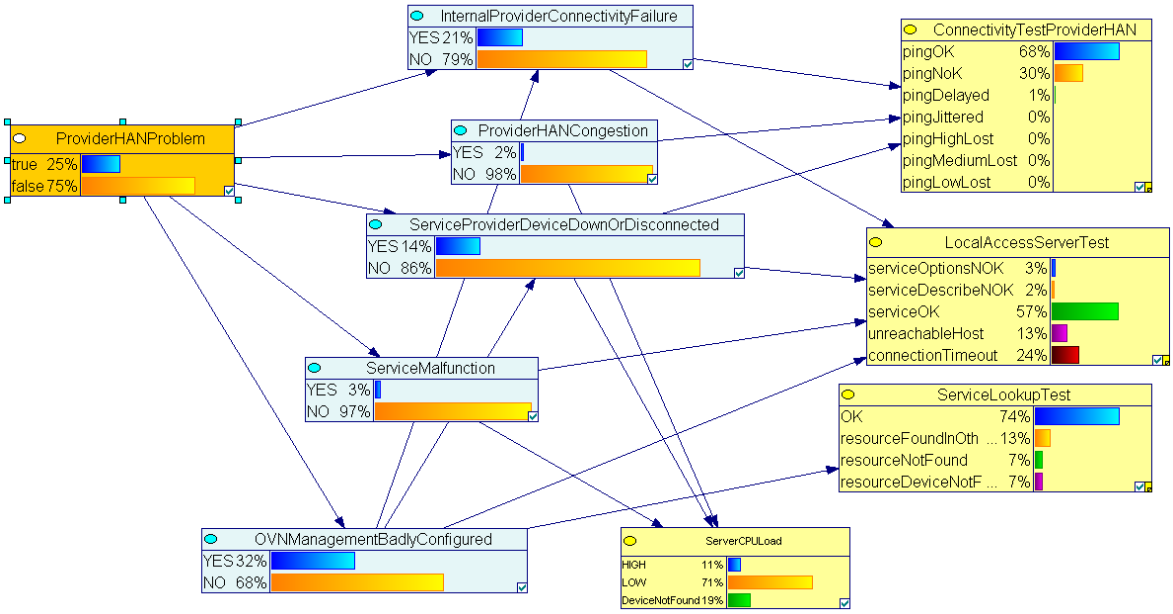
III - 4.1.b. Comparison: Fuzzy Logic vs. Bayesian Reasoning

When dealing with uncertainty, there are several options to build computational models. We have chosen two of them, which we consider the most relevant ones. Concretely we would like to compare two of these models: Bayesian Networks and Rule-Based Expert Systems with uncertainty factors. In our work, we have developed over Bayesian Network model. That is why we would like to show if a Rule-Based Expert System implementation would be viable.

As platforms to implement those models, in this comparison, we choose Genie & Smile as a Bayesian Network Inference Engine due to its beautiful and easy graphic interface and Fuzzy-Clips as a Rule-Based Expert System with uncertainty factors, due to it is easy-to-use.

In the case study presented in section VI - we have developed several Bayesian Networks that work and solve mainly the problem.

Now, we would like to see if an implementation of this system by using a Rule–Based Expert System. To make a comparison, we have implemented a little part of this Bayesian Network in a FuzzyClips code. This Bayesian Network is shown in Figure III-3.



III-3 Bayesian Network for the comparison between Fuzzy clips and Bayesian networks

This is only a part of a bigger Bayesian Network, and only a part of the network showed will be develop in FuzzyClips code. Only the nodes with two states will be included in our code because, as we see below, the work needed to implement nodes with more states grows exponentially. For each node's parent possible state, we have to add a FuzzyClips rule. Then, when asserting facts, inference will be performed.

The developed FuzzyClips code is showed below (Code III-1). Resulting facts from the inference for the facts inserted at the end of the code, are shown in Code III-2. Finally, the equivalent resulting Bayesian Network after perform inference with the same evidence (fact) is shown in Figure III-4.

Code III-1 Fuzzy Clips vs Bayesian Networks comparison

FuzzyClips code

```
;-------------------------------------------------------------
; ProviderHANCluster.clp
; Model for Magneto Network
; Created by Jesús López Méndez
; 10-Mar-2011
;-------------------------------------------------------------

(defrule InternalProviderConnectivityFailureYY
      (ProviderHANProblem yes)
      (OVNManagementBadlyConfigured yes)
=>
      (assert (InternalProviderConnectivityFailure yes) CF 0.6)
)

(defrule InternalProviderConnectivityFailureYN
      (ProviderHANProblem yes)
      (OVNManagementBadlyConfigured no)
=>
      (assert (InternalProviderConnectivityFailure yes) CF 0.3)
)

(defrule InternalProviderConnectivityFailureNY
      (ProviderHANProblem no)
      (OVNManagementBadlyConfigured yes)
=>
      (assert (InternalProviderConnectivityFailure yes) CF 0.45)
)

(defrule InternalProviderConnectivityFailureNN
      (ProviderHANProblem no)
      (OVNManagementBadlyConfigured no)
=>
      (assert (InternalProviderConnectivityFailure yes) CF 0.01)
)

;-------------------------------------------------------------

(defrule OVNManagementBadlyConfiguredY
      (ProviderHANProblem yes)
=>
      (assert (OVNManagementBadlyConfigured yes) CF 0.4)
)

(defrule OVNManagementBadlyConfiguredN
      (ProviderHANProblem no)
=>
      (assert (OVNManagementBadlyConfigured yes) CF 0.3)
)

;-------------------------------------------------------------
```

```
(defrule ServiceMalfunctionY
      (ProviderHANProblem yes)
=>
      (assert (ServiceMalfunction yes) CF 0.1)
)

(defrule ServiceMalfunctionN
      (ProviderHANProblem no)
=>
      (assert (ServiceMalfunction yes) CF 0.01)
)

;-------------------------------------------------------------

(defrule ServiceProviderDeviceDownOrDisconnectedYY
      (ProviderHANProblem yes)
      (OVNManagementBadlyConfigured yes)
=>
      (assert (ServiceProviderDeviceDownOrDisconnected yes) CF 0.4)
)

(defrule ServiceProviderDeviceDownOrDisconnectedYN
      (ProviderHANProblem yes)
      (OVNManagementBadlyConfigured no)
=>
      (assert (ServiceProviderDeviceDownOrDisconnected yes) CF 0.2)
)

(defrule ServiceProviderDeviceDownOrDisconnectedNY
      (ProviderHANProblem no)
      (OVNManagementBadlyConfigured yes)
=>
      (assert (ServiceProviderDeviceDownOrDisconnected yes) CF 0.3)
)

(defrule ServiceProviderDeviceDownOrDisconnectedNN
      (ProviderHANProblem no)
      (OVNManagementBadlyConfigured no)
=>
      (assert (ServiceProviderDeviceDownOrDisconnected yes) CF 0.01)
)

;-------------------------------------------------------------

(defrule ProviderHANCongestionY
      (ProviderHANProblem yes)
=>
      (assert (ProviderHANCongestion yes) CF 0.05)
)

(defrule ProviderHANCongestionN
      (ProviderHANProblem no)
```

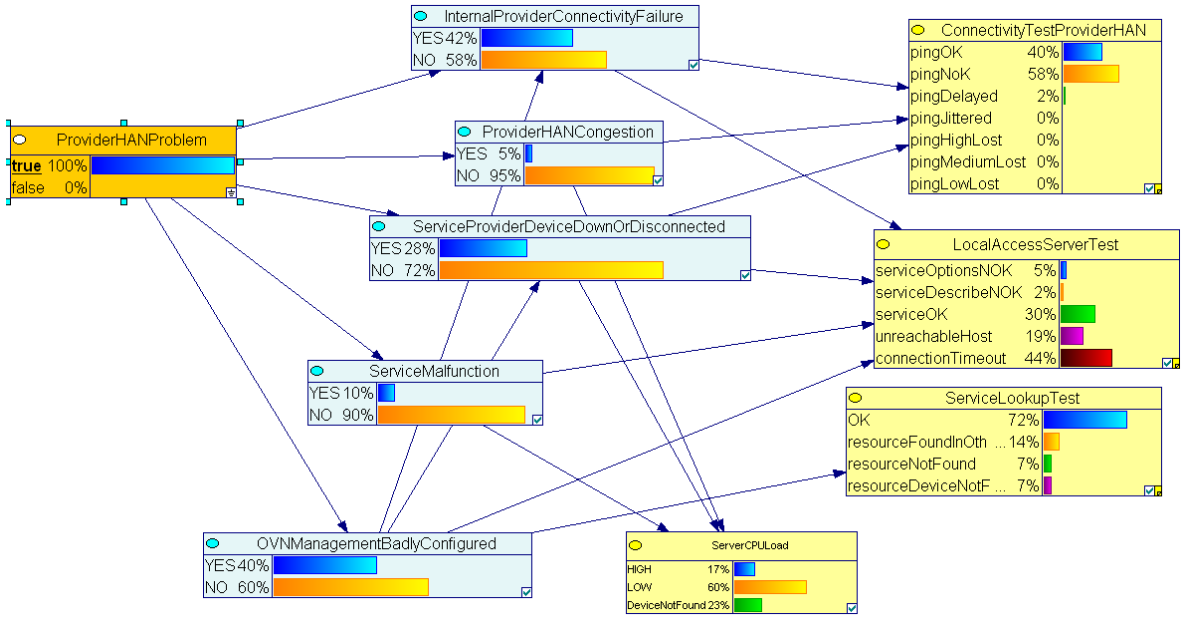
```
=>
       (assert (ProviderHANCongestion yes) CF 0.01)
)


;---------------------------------------------------------------
; FACT ASSERTION
;---------------------------------------------------------------
(deffacts UserEvidences
       (ProviderHANProblem yes) CF 1.0
)
```

Code III-2 Fuzzy Clips vs Bayesian Networks comparison results

```
Result of execution

;---------------------------------------------------------------
f-0     (initial-fact) CF 1.00
f-1     (ProviderHANProblem yes) CF 1.00
f-2     (OVNManagementBadlyConfigured yes) CF 0.40
f-3     (InternalProviderConnectivityFailure yes) CF 0.24
f-4     (ServiceProviderDeviceDownOrDisconnected yes) CF 0.16
f-5     (ServiceMalfunction yes) CF 0.10
f-6     (ProviderHANCongestion yes) CF 0.05
;---------------------------------------------------------------
```



III-4 Bayesian Network for the comparison between Fuzzy clips and Bayesian networks. Inference results with an evidence introduced

Result Analysis: Complexity Degree

As we can infer from the code, the complexity of a node codification is as follow.

Let node A be a node in the Bayesian Network representation, and let $B_1, \dots, B_n$ be the parent of node A. Then, a parent B_i has the states $S_1, \dots, S_{m_i}$. For each possible parent state, we have to add a rule to our Rule-Based System. Thus, the necessary number of rules for node A to define a similar behavior than in the Bayesian Network is as showed in Equation Eq. III-1.

$$NecessaryNumberOfRules(A) = \prod_{1}^{n} m_i \qquad \textbf{Eq. III-1}$$

This means that the number of rules needed to perform a node behavior grows exponentially with the number of its parent states. Thus, implementation of systems with moderate amounts of states per node could be viable. When the number of states per node grows a little, implementation of Rule-Based systems with uncertainty factors turns difficult and, practically, non-viable. On the other hand, Bayesian Network representation makes our job easier. Graphic interfaces allow build the system by a more intuitive way and have a clear view of system performance. Bayesian Networks are more understandable for humans and more easily developed, implemented and modified.

Result Analysis: Differences between both models

According to the results obtained in Code III-2 and in Figure III-4, some differences between both systems performance can be observed. When implementing inference between nodes directly related to the evidence node, no differences are shown. However, when there is a node between evidence and destination nodes, the results vary depending on what method. This is not an error. Simply, methods are based on different theories.

Bayesian Networks, as described in section II - 2.4, are based on Bayes theory. That means that the probabilities are computed according to conditional probability tables of the nodes. Therefore, results do not show as much the uncertainty of the information obtained but the best information we can obtain with the provided evidences.

Contrary to BN model, FuzzyClips inference engine use products to calculate the level of uncertainty. Accordingly, the degree of uncertainty will be increasing increasingly as we move away from the evidence node.

FuzzyClips describe facts and rules reality level by means of Certainty Factors (CF). Certainty Factor of a fact which is inferred (it is not an evidence) will be calculated as the product of the CF of the rule multiplying by the CF of each fact on which this rule depends.

Certainty Factors of FuzzyClips model both the uncertainty and the imprecision of facts and rules applied. In our case, we have not used uncertain / imprecise rules because we are completely sure about the relations between nodes. In fact, relations between nodes are completely precise, so the only information we have to add is their uncertainty degree.

Let us understand this better through several examples showed in the results.

Firstly, ProviderHANCongestion node shows the same numbers in both models. That is because there is not any node between this node and evidence node. In the Bayesian Network, the conditional probability presents linking this node and evidence node is $P(ProviderHANCongestionjProviderHANProblem) = 0.05$. Similarly, in the FuzzyClips model, the CF of the fact asserted by corresponding rule is also 0.05 and the CF of the previous fact (the evidence) is 1.0, so the result is the product $CF(ProviderHANCongestion) = 0.05 * 1.0 = 0.05$. As we see, when there is not any node between a node and the evidence node, obtained values are the same.

Nevertheless, InternalProviderConnectivityFailure node does not show the same results in both models. In the Bayesian Network, Bayes theorem will be applied, resulting a 0.42 probability. However, inf FuzzyClips model the CF of the fact asserted by corresponding rule is 0:6 and the CF of the previous facts are 1.0 for the evidence node and 0.4 for the OVNManagementBadlyConfigured node, resulting $CF(InternalProviderConnectivityFailure) = 0.6 * 1.0 * 0.4 = 0.24$. Thus, we see that when the distance from evidence node to a node is more than one, then the CF decrease a lot due to the lack of certainty.

About different choices shown above, we cannot simply state our preference about one of them. Each one has its strengths and its weaknesses. Bayesian Networks have many advantages, such as they are easy to build and intuitive. Bayesian Networks have a very efficient computation thanks to algorithms, which are improving more and more. In addition, they are easily distributable, what is the issue of our project. Rule–Based expert systems with uncertainty factors and, concretely, FuzzyClips, have many advantages such as they are more customizable and adaptable to our needs. We can make it more intelligent and more helpful. However, they have a more difficult implementation and modification, as well as they are hardly distributable.

III - 4.1.c. Our choice: Bayesian Reasoning

We have chosen Bayesian Networks as the reasoning technique for this project due to several reasons. Bayesian Networks have many advantages, such as they are easy to build and intuitive. In addition, Bayesian Networks have a very efficient computation thanks to algorithms, which are improving more and more. Moreover, they are easily distributable, what is the issue of our project. Other reasoning techniques shown in section II - 2 have advantages such as they are more customizable and adaptable to our human reasoning. However, they have a more difficult implementation and modification, as well as they are hardly distributable. On the other hand, Bayesian Networks are easily distributable and very adequate to incomplete knowledge, dealing easily with uncertainty.

III - 4.2. Distributed Reasoning: MSBN, our choice

Among the distributed reasoning techniques presented in II - 2, there are two which are more adequate to our problem.

DPNs are relatively simple to be implemented. The connections among different agents can be established at run time (self-configuration). It does not need prior compilation and communications are established only when necessary, thus it has high efficiency for one query variable at one time of a network with simple structure. However, DPNs has three strict

requirements for its structures as listed before, therefore, it is only suitable for some special applications.

In order to represent cooperative multi-agents who must reason with uncertain knowledge, a coherent framework is necessary. We choose multiply sectioned Bayesian networks (MSBNs) as the basis for this project because they are based on well-established theory on Bayesian networks and because they are modular.

A MSBN consists of a set of interrelated Bayesian subnetworks each of which encodes certain knowledge on a subdomain. Bayesian subnetworks are organized into a Hypertree structure such that inference can be performed in a distributed fashion while answers to queries are exact with respect to probability theory.

Each subnetwork only exchanges information with adjacent subnetworks on the Hypertree, and each pair of adjacent subnetworks only exchanges information on a set of shared variables. That is the great advantage of this organization: the complexity of communication among all agents is linear on the number of agents and the complexity of local inference is the same as if the subnet is a single agent based BN.

MSBN organization offer a simple communication model that let the system be scalable as well as reliable [20].

III - 4.3. BN Inference Frameworks: UnBBayes, our choice

Each one of the Inference Engines showed in section II - 4.5 has its own different characteristics. Hence, there is no a preferred Inference Engine. Each one has its own strengths and weaknesses.

We prefer the graphical interface of Genie, the versatility of SamIam and the readability of UnBBayes. Netica is not even an option for our project due to its commercial and we do not like to pay for it. In addition, file formats are not compatible among different inference engines as it is been shown in section II - 4.5.g. Hence, although conversion between different file formats is not too difficult, we have to make a choice.

To clarify this point, the most relevant characteristics of each Inference Engine are shown in table Table III-31.

Inference Engine	Variety of Inference Algorithms	Programming language	Maintenance	Data Formats	License
SamIam	High	Java (not totally)	High	.net (full supported), .dsl, .xdsl, .dsc, .dne, .erg	Free
UnBBayes	Low	Java	High	.net, .xml (XMLBIF)	Open Source
Genie & Smile	High	C++ (Wrapper Java)	High	.xdsl, .dsl, .erg, .dne, .dsc, .net, .dxp	Free
Netica	High	C++	High	.dne, .neta, .net, .dxp, .dsc, .ergo	Commercial

Table III-31 Inference engines comparison

First, as we mentioned below, Netica is not an option for us due to it has commercial license only.

On one hand, Genie and SamIam have clearly the most attractive and usable graphic interfaces. They are easy to use as well as intuitive and simple.

On the other hand, despite UnBBayes has not a very usable graphic interface, it has a feature that make it the choice: It is Open Source.

As we need to develop new source and reuse some code already present in the inference engine, our work becomes very simpler if we can have the original code. UnBBayes only implements Hugin algorithm and this has not been optimized too much. It only can use *.net* format and a proprietary format known as XMLBIF that is not used for any other inference engine. However, the availability of its source code and the fact that it is programmed completely in java language, make it the optimal choice for this project.

According to the analysis done, **UnBBayes is chosen as the Inference Engine for our project**.

III - 4.4. Distributed Communication Frameworks: JGroups, our choice

Among the set of tools studied in section II - 6 we value JGroups as the best choice. While Hazelcast is a framework for sharing memory, JGroups is a framework for message passing.

Hazelcast has been developed to share data among servers or cache data to achieve faster response times while trying to avoid single point of failures. It has a good response to dynamic events, as node crashes but those events are transparent for user. Thus, the treatment and reaction to fails and crashes is more difficult and communication between nodes in the MSBN is not the preferred for this distributed reasoning technique.

Consequently, JGroups is chosen as the communication framework of this project since it offer great advantages such as easy managing of all active members, reliable communication between them, detection and notification about joined / left / crashed members or point-to-multipoint and point-to-point communications.

IV - Architecture and Design

In this chapter, a detailed description of the whole developed system is shown. Concretely, it is separated in two different sections, one for each developed architecture.

The first presented architecture is the synchronous one, which is an adaptation of the architecture presented in section II - 5.4 to make it available to the use in multi-agent systems.

The second presented architecture is the iterative one. This is the most important part of this project. It allows the use of this in multi-agent systems, as well as allows the reliability and robustness to crashes or errors.

IV - 1. System parts

This distributed reasoning framework consists of Hypernodes, which are the only units that compose it.

Each Hypernode is completely independent of other Hypernodes. This is the great advantage of this architecture: each Hypernode can be created or stopped without affect the correct functioning of the whole MSBN.

A Hypernode has a number of private nodes and other that are public. Public nodes are those that are shared with other Hypernodes, and are used as communication channel between several Hypernodes.

Hypernodes are connected through Linkages. In turn, Linkages are formed of Links, which are the basic unit, linking pairs of cliques. For more details about this architecture, a more detailed description is given in section II - 4.3.

IV - 2. Developed MSBN Architectures

The MSBN architecture studied in II - 5.3 is implemented to allow reasoning with uncertainty in a synchronous Single-Agent System. This departing architecture is known as Single-Agent MSBN with synchronous communications. This means that the system will run over an only agent that will contain the entire MSBN. In addition, we have to highlight that this architecture use synchronous communications. That is to say, processes are performed at the same time in the different subnetworks of the MSBN, what can result in useless waiting times.

The single-agent paradigm is inadequate when uncertain reasoning is performed by elements of a system between which there is some “distance”, which may be spatial, temporal or semantic. Such systems pose special issues that need to be addressed. A multi-agent view is thus required where each subnetwork part is an autonomous intelligent subsystem.

Although each subnetwork does not necessarily have to belong to an agent, and could be used by any program, in this project we will focus on agent paradigm to do the explanation easier.

Assuming a multi-agent paradigm, each agent holds its own partial domain knowledge, accesses some external information source and consumes some computational resource. Each agent communicates with other agents to achieve the system’s goal cooperatively. MSBNs provide a simple way of communication between agents to share only those beliefs that are public and only with those agents that require them. In addition, MSBN is a precise way that allows dealing with the lack of information.

In our work, we start from the study of the already developed synchronous Single-Agent architecture to develop other architectures more appropriate to our problem. We have developed and implemented two different architectures to allow the use of MSBNs in Multi-Agent Systems (MASs): synchronous architecture and iterative architecture.

As synchronous SAS architecture does, synchronous MAS architecture uses a synchronous and centralized architecture. Although the selection of the root/coordinator node can be done arbitrarily, all processes in the compilation of the BN and in the communication of beliefs in the BN are done synchronously. This is achieved by mean of recursive calling to the corresponding

methods in the different subnetworks. Since all methods are called recursively, while processes are performed in each subnetwork, all other subnetworks are in active waiting.

To extend single-agent MSBNs into MAS, many issues need to be resolved. First steps involve the coherent agent communication[21], the optimization of communication scheduling[22] and the distributed structure verification[23].

IV - 2.1. Synchronous Architecture for Multi-Agent MSBN

IV - 2.1.a. Introduction

Notation used in this section

As a link before this report and the programmed code, this section use some notation that correspond to programmatic language. The conventions used are:

- Names of objects are shown corresponding to programmed ones. That is, to refer to a link that belongs to a Linkage that is in the list of this subnetwork named *links*, we will use the notation *subnetwork.links.linkage.linkList.link*. The explanation of this notation is, since the subnetwork has a list of Linkages called *links*, a linkage is trivially called *linkage*, each Linkage has a list of Links called *linkList*, and each Link in this list is trivially called *link.*
- Objects used to extract some information at a determinate step or process are enclosed between curly brackets {}. For example, if in a determinate step we use the Links of a Linkage to know the shared nodes, after the description of this step *{linkage.linkList.link}* will be written.
- Objects modified during a determinate step or process are enclosed between square brackets []. For example, if in a determinate step we modify a property of a Link of a Linkage, after the description of this step *[linkage.linkList.link.property]* will be written.
- The interface methods used for communication between subnetworks are written in a gray color.

Communication Framework

The architecture needs a communication framework that supports synchronous operations. That is, a framework that supports Remote Procedure Call (RPC). It has been manually implemented by the following means:

- Direct method calling to emulate communication in an only computer.
- Socket based communication that allows the calling of methods remotely according to the message passed. In addition, some objects can be serialized to be sent.
- Jadex Agent Platform, which implements itself serializing methods for messages.

Class Diagram

To describe the developed architecture, the class diagram shown in Figure IV-1 is proposed. There the relationships between the more important classes belonging to the developed architecture can be seen. Those classes that belong to the core of UnBBayes, such as Node, Network or Edge, are not shown in this diagram due to they have not been written by us.

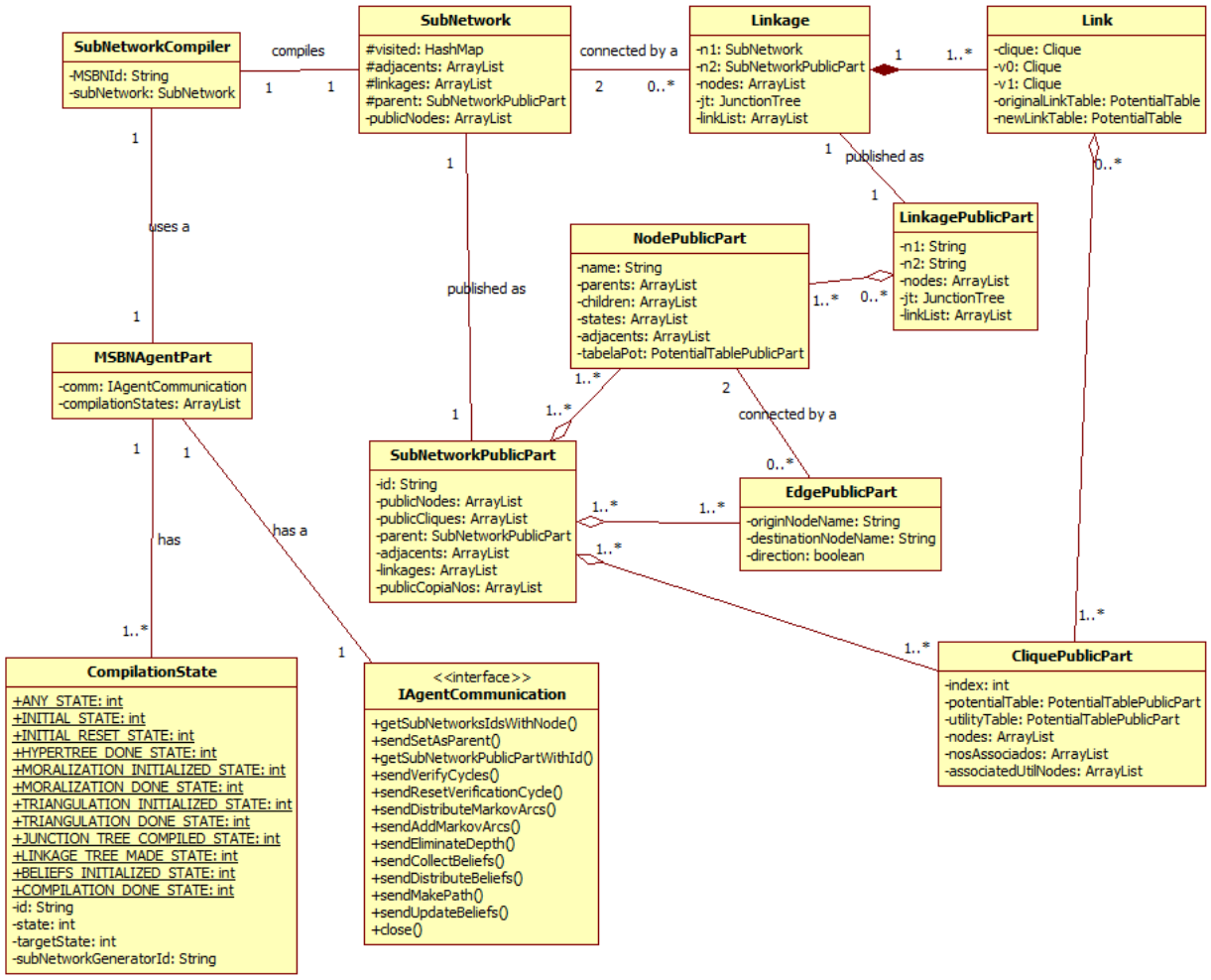


Figure IV-1 Class diagram for Synchronous architecture

State Diagram

In synchronous architecture, compilation is not able to go back at any time. Thus, the whole process need to be repeated each time there are any structure change in the MSBN. These characteristics can be observed in the state diagram shown in Figure IV-2.

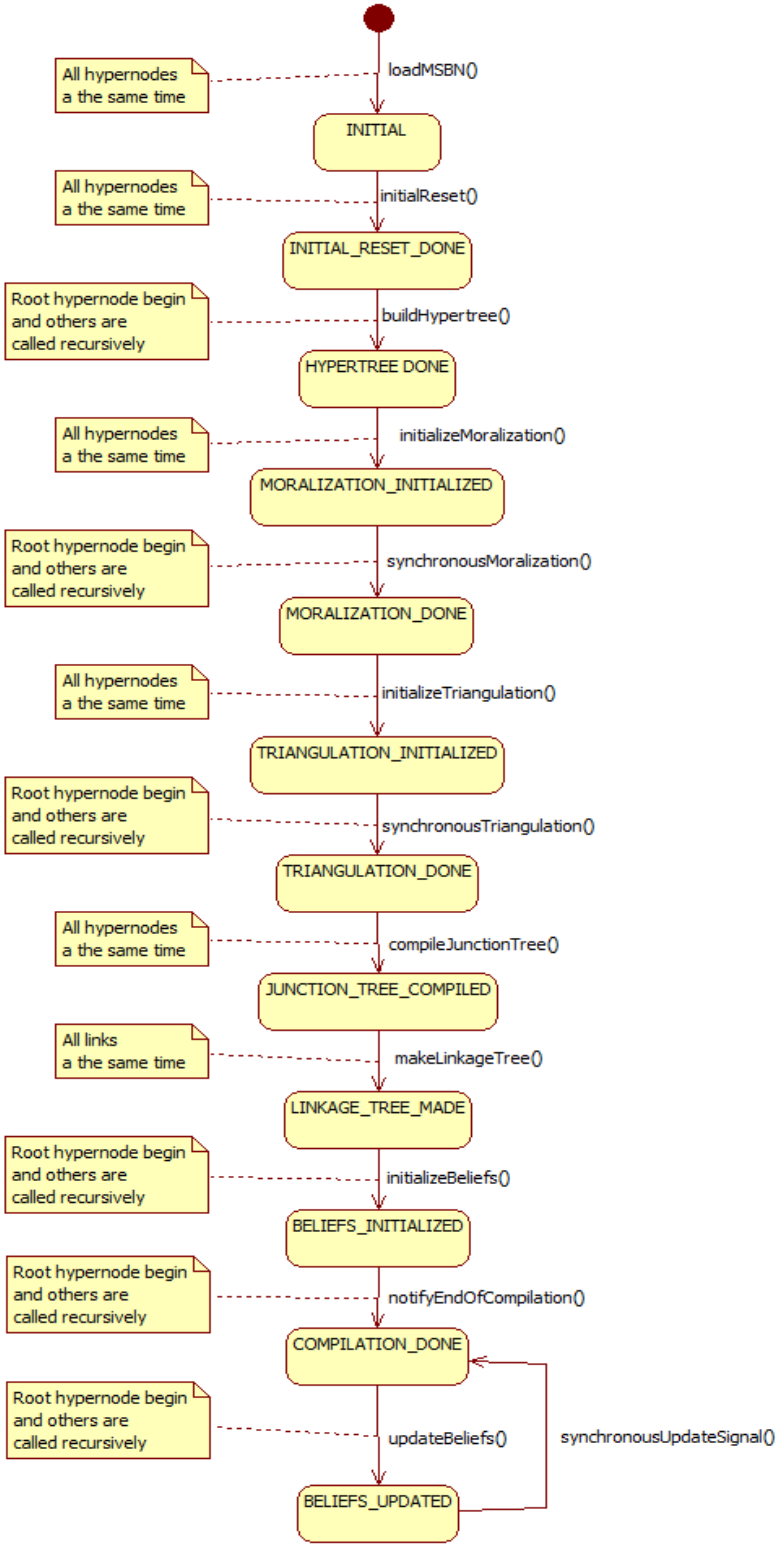


Figure IV-2 State diagram for Synchronous architecture

IV - 2.1.b. Detailed Description

In this section, we give a high detailed description of the basic operations performed along the states given above. For that purpose, we use the notation conventions given in section IV - 2.1.a.

IV - 2.1.b.1. Compilation

From ANY_STATE to INITIAL_STATE

- This compilation state is only used for indicate that no state has been set yet.

From INITIAL_STATE to INITIAL_RESET_STATE

- Clear Linkages, Adjacents and parent. [linkages, adjacents, parent]
- Verify Consistency

From INITIAL_RESET_STATE to HYPERTREE_DONE_STATE

- Find intersection this and the other subnetworks.
  getSubNetworkWithNode
  getSubNetworkPublicPart(requiredState=INITIAL_RESET)
- Choose one subnetwork as an adjacent.
- Ask him to put this subnetwork as parent.
  setAsParent [parent of the subnetwork]
- Put subnetwork chosen as adjacent of this subnetwork. [adjacent]
- Add a linkage between this and its adjacent subnetwork. This subnetwork is the owner of this linkage. [linkages]

From HYPERTREE_DONE_STATE to MORALIZATION_INITIALIZED_STATE

- Verify Cycles (in the whole MSBN).
  sendVerifyCycles
  sendResetVerificationOfCycles
- Local Moralization. Add fill-ins needed to arcosMarkov. [arcosMarkov]

From MORALIZATION_INITIALIZED_STATE to MORALIZATION_DONE_STATE

- For each adjacent:
  - Ask to perform local moralization. [ArcosMarkov]
    getSubNetworkPublicPartWithId(requiredState=MORALIZATION_INITIALIZED)
- For each adjacent:
  - Distribute arcosMarkov of this subnetwork.
    sendAddMarkovArcs
  - Ask to perform complete moralization and return arcosMarkov. [ArcosMarkov]
    getSubNetworkPublicPartWithId(requiredState=MORALIZATION_DONE)
- For the parent (if exists):
  - Distribute arcosMarkov of this subnetwork. [arcosMarkov of the parent]
    sendAddMarkovArcs

From MORALIZATION_DONE_STATE to TRIANGULATION_INITIALIZED_STATE

- Make a copy of nodeList of this subnetwork in its copiaNos. [copiaNos]
- Clear the elimination order list (oe) of this subnetwork. [oe]

From TRIANGULATION_INITIALIZED_STATE to TRIANGULATION_DONE_STATE

- Eliminate Depth (caller=null):
    - For each adjacent: [adjacent]
        - Update this adjacent *getSubNetworkPublicPartWithId(requiredState=TRIANGULATION_INITIALIZED)*
        - If(minimumWeightTriangulation(adj))
            - Distribute the arcs added during triangulation of this subnetwork. *distributeMyArcsTo(destination=adjacent)*
        - For each adjacent different than caller:
            - Ask to Eliminate Depth: *sendEliminateDepth(caller=thisSubnetwork)*
        - If(adjacent has parent)
            - If(minimumWeightTriangulation(parent))
                - Distribute the arcs added during triangulation of this subnetwork. *distributeMyArcsTo(destination=parent)*
- For each adjacent:
    - Update this adjacent *GetSubNetworkPublicPartWithId(requiredState=TRIANGULATION_INITIALIZED)*
    - Distribute the arcs added during triangulation of this subnetwork. *distributeMyArcsTo(destination=adjacent)*
    - Ask to perform Triangulation and return arcs [arcs] *getSubNetworkPublicPart(requiredState=TRIANGULATION_DONE)*
- If(this subnetwork has parent)
    - Ask to init triangulation. *getSubNetworkPublicPart(requiredState=TRIANGULATION_INITIALIZED)*
    - Distribute the arcs (added during triangulation of this subnetwork. *distributeMyArcsTo(destination=parent)*

From TRIANGULATION_DONE_STATE to JUNCTION_TREE_COMPILED_STATE

- Reset the evidences in each node of the subnetwork. [node.evidence]
- Put a new Junction Tree as jt of this subnetwork. It has separator and clique lists empty. [jt]
- Add all possible cliques of this subnetwork to jt.cliques. [jt.cliques]
- Associate an index to each clique in jt.cliques and sort jt.cliques according to this index. [jt.clique.index, jt.cliques (order)]
- Sort nodes in cliques and separators according to the elimination order. [jt.clique.nos (order)]

- For each node in this subnetwork:
    - Add to each jt.clique tables (potentialTable and utilityTable) as many variables as nodes in this clique. {jt.clique.node} [jt.clique.potentialTable, jt.clique.utilityTable]
    - Add to each jt.separator tables (potentialTable and utilityTable) as many variables as nodes in the clique. {jt.separator.node} [jt.separator.potentialTable, jt.separator.utilityTable]
    - For each node in the subnetwork, add this node to the appropriate list of the jt.clique which has the minimum size potentialTable. This appropriate list is nosAssociados in the case of this node is a ProbabilisticNode or associatedUtilNodes otherwise. {nodeList.node}[jt.clique.nosAssociados, jt.clique.associateUtilNodes]
- Init the beliefs of the jt:
    - If the beliefs of the jt haven't been initialized yet:
        - For each clique in jt:
            - Set potentialTable values to 1. [jt.clique.potentialTable]
            - Multiply the potentialTable by each nosAssociados.node potentialTable. {jt.clique.nosAssociados.node.potentialTable}[jt.clique.potentialTable]
            - Set utilityTable values to 0. [jt.clique.utilityTable]
            - Add to the utilityTable each associatedUtilNodes.node utilityTable. {jt.clique.associatedUnitNodes.node.utilityTable}[jt.clique.utilityTable]
        - For each separator in jt:
            - Set potentialTable values to 1. [jt.separator.potentialTable]
            - Set utilityTable values to 0. [jt.separator.utilityTable]
        - Make consistent by collecting and distributing evidences.
        - Make a internal copy of the potentialTable and utilityTable of all cliques and separators in jt.{jt.clique.potentialTable.dataPT, jt.clique.utilityTable.dataPT, jt.separator.potentialTable.dataPT, jt.separator.utilityTable.dataPT} [jt.clique.potentialTable.dataCopy, jt.clique.utilityTable.dataCopy, jt.separator.potentialTable.dataCopy, jt.separator.utilityTable.dataCopy]
    - If the beliefs of the jt have already been initialized:
        - Restore data from the internal copy done. {jt.clique.potentialTable.dataCopy, jt.clique.utilityTable.dataCopy, jt.separator.potentialTable.dataCopy, jt.separator.utilityTable.dataCopy} [jt.clique.potentialTable.dataPT, jt.clique.utilityTable.dataPT, jt.separator.potentialTable.dataPT, jt.separator.utilityTable.dataPT]
    - Make consistency by mean of collect and distribute evidences. {jt.clique.child} [jt.clique.potentialTable, jt.separator.potentialTable]

- Make a copy of the data in the tables: copy both potentialTable and utilityTable from dataPT to dataCopy in all cliques and separators.
- For each node in copiaNos:
  {jt.sep.potentialTable, jt.clique.potentialTable}[node.associatedClique]
  - If it's a ProbabilisticNode: look for the separator that contains this node which has the smallest potentialTable and set this separator as associatedClique of the node.
  - If it's a DecisionNode or it doesn't exist a separator which contains this node: look for the clique that contains this node which has the smallest potentialTable and set this clique as associatedClique of the node.
- For each node in copiaNos:
  - Init marginalList as a new array of Floats. [node.marginalList]
  - Set marginalList values to values obtained from nodo.cliqueAssociado.potentialTable.
    {node.cliqueAssociado.potentialTable} [node.marginalList]

From JUNCTION_TREE_COMPILED_STATE to LINKAGE_TREE_MADE_STATE

For each linkage:

- Clear the linkList. [linkage.linkList]
- Assign a new jt (junction tree) to this linkage. [linkage.jt]
- Call makeCliqueList(n1.jt.clique0) method, where makeCliqueList(Clique c) does the following:
  {n1.jt.cliques}
  - Create a new clique (b) with the nodes intersection between this linkage and c.
  - Add b to jt.cliques. [linkage.jt.clique]
  - Add to jt.linkList a new link with b as host0. [linkage.jt.linkList]
  - For each clique child of c:
    [linkage.jt.clique, linkage.jt.linkList, linkage.jt.clique.parent, linkage.jt.clique.child]
    - Call makeCliqueList(child) obtaining b2, the new intersection clique created from the child.
    - Set b as parent of the child.
    - Add b2 as child of b.
  - Return b.
- Call remove1stPass method where, for each link in linkage.jt.linkList, the following is done:
  [linkage.jt.linkList]
  - If the clique hasn’t children:
    - Remove nodes from linkage.jt.linkList.link.clique that are already in the parent of this clique.
    - If the clique hasn’t nodes, remove this link.

- Call remove2ndPass method where, for each link in linkage.jt.linkList, the following is done:
  [linkage.jt.linkList]
  - If all nodes in link.clique are already in the parent: remove link.
  - If any of the children of the parent of this clique (a brother) has all of its nodes: remove link.
- Call assignV1 method where, for each link in linkage.jt.linkList, the following is done:
  [linkList.link.Host1]
  - Look for the linkage.n2.jt.cliques.clique what contains all the nodes of the link.clique and set this as link.Host1.
- Call initTables method, where the following is done:
  [jt.separators, linkList.link.clique.potentialTable, jt.separators.separator.potentialTable]
  - InitSeparators: for each linkage.jt.clique:
    - For each clique.child:
      - Construct a new separator with the nodes of the intersection of its nodes.
  - For each linkList.link.clique.potentialTable:
    - Add as many variables as nodes in the clique.
    - Set all of these variables to 1.
  - For each jt.separators.separator.potentialTable:
    - Add as many variables as nodes in the separator.
    - Set all of these variables to 1.

From LINKAGE_TREE_MADE_STATE to BELIEFS_INITIALIZED_STATE

Starting in the subnetwork net0:

- Call collectBeliefs(net0) method where, the following is done:
  - For each adjacent (netAdj):
    - If netAdj has adjacents, call collectBeliefs(netAdj)
      sendCollectBeliefs(destinationSubNetworkId, requiredCompilationState)
    - Call updateBeliefs (net, ← netAdj)
      sendUpdateBeliefs(destinationSubNetworkId, fromId, fromAdjacent)
- Call distributeBeliefs(net0) method where, the following is done:
  - For each adjacent (netAdj):
    - Call updateBeliefs(netAdj, ← net)
      sendUpdateBeliefs(destinationSubNetworkId, fromId, fromAdjacent)
    - Call distributeBeliefs(netAdj)
      sendDistributeBeliefs(destinationSubNetworkId, requiredCompilationState)

updateBeliefs(netToUpdate, fromNet):

- For each links.linkage:
  - Linkage.absorb(fromAdjacent = true)
    - For each linkage.linkList.link:
      - Link.absorbIn(fromAdjacent) [link.originalLinkTable, link.newLinkTable, link.clique.potentialTable]
    - RemoveRedundancy:
      - For each linkage.jt.separator:
        - Remove from separator.probabilityFunction all variables which are in linkage.clique2 and are not in separator.
        - Copy to the separator.probabilityFunction values in linkage.clique2.
      - For each linkList.link which clique is linkage.clique2: link.removeRedundancy()
    - For each linkage.linkList.link:
      - Link.absorbOut(fromAdjacent) [link.originalLinkTable, link.newLinkTable, link.v0/v1.potentialTable]
    - For the subnetwork to what beliefs propagate:
      - Absorb2()

Link.absorbIn(fromAdjacent):

- Copy clique.potentialTable to originalLinkTable {link.clique.potentialTable}[link.originalLinkTable]
- Copy the part of clique (fromAdjacent / fromParent) link.v1/v0.potentialTable whose nodes are in link.clique, to link.newLinkTable. {link.v1/v0.potentialTable}[link.newLinkTable]
- Copy values in newLinkTable to link.clique.PotentialTable. {link.newLinkTable}[link.clique.potentialTable]

In short, take the corresponding part of clique link.v1/v2 and set it as link.clique.potentialTable.

Link.absorbOut(fromAdjacent):

- Divide link.newLinkTable by originalLinkTable. {link.originalLinkTable}[link.newLinkTable]
- Multiply link.v0/v1.potentialTable by link.newLinkTable.potentialTable obtained. {link.newLinkTable}[link.v0/v1.potentialTable]

SubNetwork.Absorb2():

This.jt.consistency()

This.updateMarginals() [nodeList.node.marginalList]

From BELIEFS_INITIALIZED_STATE to COMPILATION_DONE_STATE

Do nothing. This compilation will represent the complete update of beliefs in the iterative version presented in the following section.

IV - 2.1.b.2. Adding and propagating Beliefs

Add Finding

- Set the evidence of this node to the number passed. [node.evidence]
- Set the marginal of this node according to the evidence set. {node.evidence} [node.marginalList]

Update Evidences (at SubNetwork class)

- For each node with evidences, call to Update Evidences (at TreeVariable class):
    - Multiply the node.associatedClique.potentialTable.dataPT by marginals in node.marginalList. {node.marginalList} [node.associatedClique.potentialTable.dataPT]
- Make the consistency of this subnetwork.junctionTree:
    - Update junctionTree.n with the normalize probability.
    - Collect evidences to the root node by updating the separator.potentialTable and the clique.potentialTable {clique2.potentialTable} [separator.potentialTable, clique1.potentialTable]
    - Distribute evidences from the root node by updating the separator.potentialTable and the clique.potentialTable {clique1.potentialTable} [separator.potentialTable, clique2.potentialTable]
- For each node which is a TreeVariable, call to UpdateMarginals (at ProbabilisticNode class):
    - Update node.marginalList with values obtained from node.associatedClique.potentialTable. {node.associatedClique.potentialTable} [node.marginalList].
- For each node if node.hasLikelihood property is true, remove evidences from this node and set hasLikelihood to false. {node.hasLikelihood} [node.evidence, node.hasLikelihood].

Shift Attention from a subnetwork to another

- Make path: obtain a list with the subnetworks, which need to be updated to update a concrete subnetwork.
- For each pair of linked subnetwork composing the path, following the path, call to UpdateBelief. This is described above, in Compilation Process.

IV - 2.2. Iterative Architecture for Multi-Agent MSBN

IV - 2.2.a. Introduction

This architecture proposes that each subnetwork (or its associated agent) can work independently. Even though multiple agents may acquire evidence asynchronously in parallel (compare with the single user case where evidence is always entered into the current subnet), the corresponding communication operations of MSBNs ensure that the answers to queries

from each agent are consistent with evidence acquired in the entire system after each communication. Since communication is infrequent, the operations also ensure that between two successive communications, the answers to queries for each agent are consistent with all local evidence gathered so far and are consistent with all evidence gathered in the entire system up to the last communication. Therefore, this architecture can be characterized as one of functionally accurate, cooperative distributed systems.

Notation used in this section

As a link before this report and the programmed code, this section uses some notation that corresponds to programmatic language. The conventions used are:

- Names of objects are shown corresponding to programmed ones. That is, to refer to a link that belongs to a Linkage that is in the list of this subnetwork named *links*, we will use the notation *subnetwork.links.linkage.linkList.link*. The explanation of this notation is, since the subnetwork has a list of Linkages called *links*, a linkage is trivially called *linkage*, each Linkage has a list of Links called *linkList*, and each Link in this list is trivially called *link.*
- Objects used to extract some information at a determinate step or process are enclosed between curly brackets {}. For example, if in a determinate step we use the Links of a Linkage to know the shared nodes, after the description of this step *{linkage.linkList.link}* will be written.
- Objects modified during a determinate step or process are enclosed between square brackets []. For example, if in a determinate step we modify a property of a Link of a Linkage, after the description of this step *[linkage.linkList.link.property]* will be written.
- The interface methods used for communication between subnetworks are written in a gray color.

Communication Framework

This architecture uses a communication framework that supports asynchronous message sending. This means that, when a message arrives, a simple operation (normally queue the information conveniently) needs to be done. It has been achieved using an already developed communication framework: JGroups, which is a reliable group communication toolkit written entirely in Java. It is based on IP multicast.

Although developed architecture is independent from communication framework, there are several services provided by JGroups toolkit that are not easily found in other communication frameworks, such as:

- Notification about joined / left /crashed members
- Point-to-multipoint and Point-to-point messaging
- State transmission

Class Diagram

To describe the developed architecture, the class diagram shown in Figure IV-3 is proposed. There the relationships between the more important classes belonging to the developed architecture can be seen. Those classes that belong to the core of UnBBayes, such as Node, Network or Edge, are not shown in this diagram due to they have not been written by us.

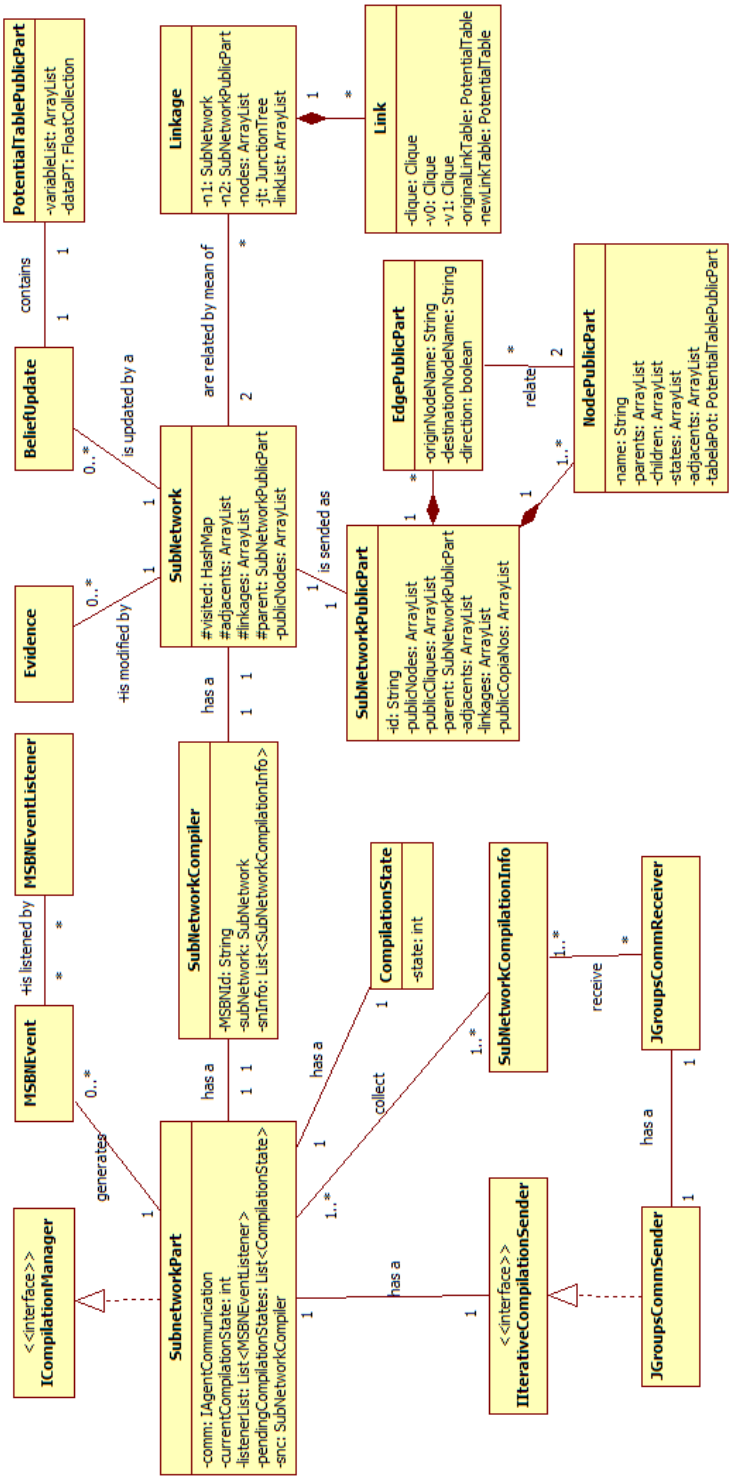


Figure IV-3 Class diagram for Iterative Architecture

State Diagram

In Iterative architecture, the state of the compilation can go back to a previous state. This can contrast with the previous state logic shown in section IV - 2.1.a for synchronous architecture. State diagram shown in Figure IV-4 includes those states that belong to compilation process, as well as those that precede and follow it.

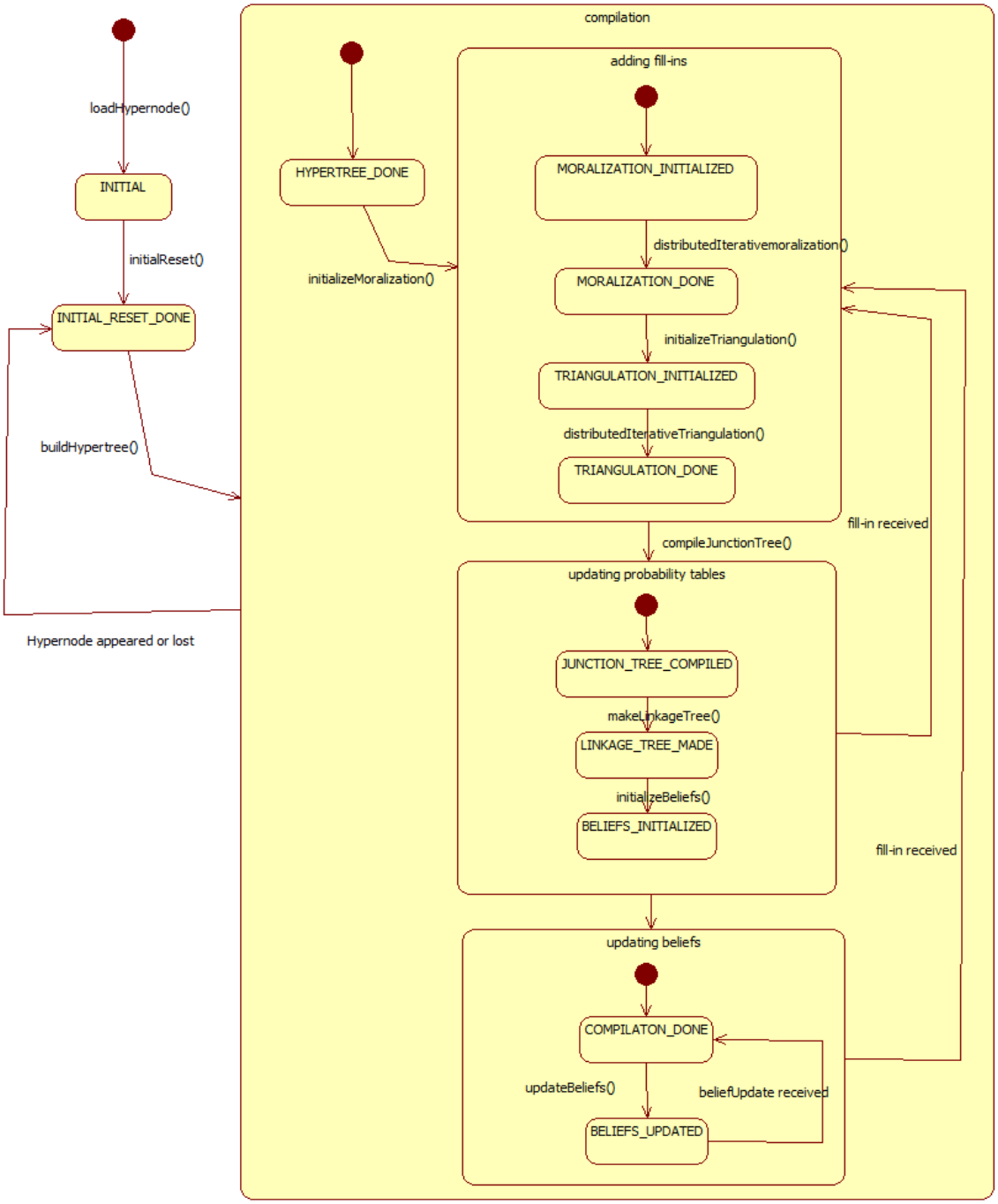


Figure IV-4 State diagram for Iterative architecture

IV - 2.2.b. Detailed Description

In this section, we give a high detailed description of the basic operations performed along the states given above. For that purpose, we use the notation conventions given in section IV - 2.1.a.

IV - 2.2.b.1. Initialization

Create a new SubNetworkPart

- Associate subnetwork file to load.
- Associate an even listener.

Start

- Create a new Thread (loaderThread) which will call startInTheSameThread.

StartInTheSameThread

- Load the SubNetworkCompiler (*snc*) associated to the corresponding file. This snc will have already loaded the subnetwork.
- Set the initial currentCompilationState to ANY_STATE and the targetCompilationState to COMPILATION_DONE_STATE.
- Clear pendingCompilationStates list.
- Create a new JGroupsCommSender and a new JGroupsCommReceiver.
- Start a new Thread (compilationThread), which will try to compile repeatedly while *keepAlive* is true.

IV - 2.2.b.2. Compilation

From ANY_STATE to INITIAL_STATE

- This compilation state is only used for indicate that no state has been set yet.

From INITIAL_STATE to INITIAL_RESET_STATE

- Clear Linkages, Adjacents and parent. [linkages, adjacents, parent]
- Verify Consistency
- Local Moralization. Add fill-ins needed to *arcosMarkov*. Make a copy of *edgeList* in *copiaArcos*, but removing all edge with destination in a decision node. {edgeList} [copiaArcos, arcosMarkov]
- Only the first time:
    - Save the public info about this subnetwork (*snInfo*) in the *snInfo* sorted map. [snInfo]
    - Send the public info about this subnetwork to all Hypernodes in MSBN. *publishNodes*

From INITIAL_RESET_STATE to HYPERTREE_DONE_STATE

- CheckLinks:
    - Find the intersection between each pair of subnetworks with the information contained in *snInfo*.
    - Choose as root the node that is the first in the sorted list *snInfo*, sorting the items according to alphabetical order.
    - Recursively, assign linkages between the subnetwork that is not in hypertree yet and has the largest amount of nodes in common with another subnet that is already in hypertree.
    - If there are any networks that cannot be connected with hypertree, another root node will be chose to continue this process. In this case, our MSBN is divided in several MSBN due to some subnetwork is down.
    - If there are any changes in parent or adjacent subnetworks, these changes are applied and continue compilation will be necessary. If, on the contrary, there are no changes in this part of hypertree, we will jump to LINKAGE_TREE_MADE_STATE compilation state.
- Verify Cycles (in the whole MSBN):

    Root Hypernode will send a message to call all subnetworks to start marking its nodes. This message has an associated cycleTestingId. Only last received id will be used to check cycles.
    sendVerifyCycles

    When each subnetwork receives this message, will start to mark all leaf and root nodes that it has. In the case of public nodes, only will be marked those root nodes whose parents are contained in this subnetwork and all of them are already marked. A public leaf node will be marked when all children of this node have been marked in each subnetwork. For this purpose, a subnetwork will send a message to those that share the same node when it has marked all the children of that node.
    sendNotifyMarkedChildNodes

    When every public node is marked, a message will be sent to all subnetworks that share this node.
    SendDistributeMark

    Finally, when a subnetwork has all its nodes marked, will send a message to the root to indicate that. Root will store these messages to know when the MSBN has completed the cycle check. Then, root will send the same message to all subnetworks. The meaning of this message depends on who is the sender.
    sendNotifyCycleVerificationDone

From HYPERTREE_DONE_STATE to MORALIZATION_INITIALIZED_STATE

- Restore the original markov arcs obtained in local moralization.

From MORALIZATION_INITIALIZED_STATE to MORALIZATION_DONE_STATE

- For each adjacent:

  Ask to perform local moralization. [ArcosMarkov]
  getSubNetworkPublicPartWithId(requiredState=MORALIZATION_INITIALIZED)

- For each adjacent:

  Distribute arcosMarkov of this subnetwork.
  sendAddMarkovArcsFromMoralization

- For the parent (if exists):

  Distribute arcosMarkov of this subnetwork. [arcosMarkov of the parent]
  sendAddMarkovArcsFromMoralization

From MORALIZATION_DONE_STATE to TRIANGULATION_INITIALIZED_STATE

- Make a copy of nodeList of this subnetwork in its copiaNos. [copiaNos]
- Clear the elimination order list (oe) of this subnetwork. [oe]

From TRIANGULATION_INITIALIZED_STATE to TRIANGULATION_DONE_STATE

- Eliminate Depth (caller=null):
  - For each adjacent: [adjacent]
    - Update this adjacent

      getSubNetworkPublicPartWithId(requiredState=TRIANGULATION_INITIALIZED)

    - If(minimumWeightTriangulation(adj))

      Distribute the arcs added during triangulation of this subnetwork.
      distributeMyArcsTo(destination=adjacent)

    - For each adjacent different than caller:

      Ask to Eliminate Depth:
      sendEliminateDepth(caller=thisSubnetwork)

    - If(adjacent has parent)
      - If(minimumWeightTriangulation(parent))
        - Distribute the arcs added during triangulation of this subnetwork.

          distributeMyArcsTo(destination=parent)

- For each adjacent:

  Update this adjacent
  GetSubNetworkPublicPartWithId(requiredState=TRIANGULATION_INITIALIZED)

Distribute the arcs added during triangulation of this subnetwork.
distributeMyArcsTo(destination=adjacent)

Ask to perform Triangulation and return arcs [arcs]
getSubNetworkPublicPart(requiredState=TRIANGULATION_DONE)

- If(this subnetwork has parent)

  Ask to init triangulation.
  getSubNetworkPublicPart(requiredState=TRIANGULATION_INITIALIZED)

  Distribute the arcs added during triangulation of this subnetwork.
  distributeMyArcsTo(destination=parent)

From TRIANGULATION_DONE_STATE to JUNCTION_TREE_COMPILED_STATE

- Reset the evidences in each node of the subnetwork. [node.evidence]
- Put a new Junction Tree as jt of this subnetwork. It has separator and clique lists empty. [jt]
- Add all possible cliques of this subnetwork to jt.cliques. [jt.cliques]
- Associate an index to each clique in jt.cliques and sort jt.cliques according to this index. [jt.clique.index, jt.cliques (order)]
- Sort nodes in cliques and separators according to the elimination order. [jt.clique.nos (order)]
- For each node in this subnetwork:
  - Add to each jt.clique tables (potentialTable and utilityTable) as many variables as nodes in this clique. {jt.clique.node} [jt.clique.potentialTable, jt.clique.utilityTable]
  - Add to each jt.separator tables (potentialTable and utilityTable) as many variables as nodes in the clique. {jt.separator.node} [jt.separator.potentialTable, jt.separator.utilityTable]
  - For each node in the subnetwork, add this node to the appropriate list of the jt.clique which has the minimum size potentialTable. This appropriate list is nosAssociados in the case of this node is a ProbabilisticNode or associatedUtilNodes otherwise. {nodeList.node}[jt.clique.nosAssociados, jt.clique.associateUtilNodes]
- Init the beliefs of the jt:
  - If the beliefs of the jt haven't been initialized yet:
    - For each clique in jt:
      - Set potentialTable values to 1. [jt.clique.potentialTable]
      - Multiply the potentialTable by each nosAssociados.node potentialTable.

{jt.clique.nosAssociados.node.potentialTable}[jt.clique.potentialTable]

- Set utilityTable values to 0. [jt.clique.utilityTable]
- Add to the utilityTable each associatedUtilNodes.node utilityTable. {jt.clique.associatedUnitNodes.node.utilityTable}[jt.clique.utilityTable]

▪ For each separator in jt:

- Set potentialTable values to 1. [jt.separator.potentialTable]
- Set utilityTable values to 0. [jt.separator.utilityTable]

▪ Make consistent by collecting and distributing evidences.

▪ Make a internal copy of the potentialTable and utilityTable of all cliques and separators in jt.{jt.clique.potentialTable.dataPT, jt.clique.utilityTable.dataPT, jt.separator.potentialTable.dataPT, jt.separator.utilityTable.dataPT} [jt.clique.potentialTable.dataCopy, jt.clique.utilityTable.dataCopy, jt.separator.potentialTable.dataCopy, jt.separator.utilityTable.dataCopy]

o If the beliefs of the jt have already been initialized:

▪ Restore data from the internal copy done. {jt.clique.potentialTable.dataCopy, jt.clique.utilityTable.dataCopy, jt.separator.potentialTable.dataCopy, jt.separator.utilityTable.dataCopy} [jt.clique.potentialTable.dataPT, jt.clique.utilityTable.dataPT, jt.separator.potentialTable.dataPT, jt.separator.utilityTable.dataPT]

o Make consistency by mean of collect and distribute evidences. {jt.clique.child} [jt.clique.potentialTable, jt.separator.potentialTable]

o Make a copy of the data in the tables: copy both potentialTable and utilityTable from dataPT to dataCopy in all cliques and separators.

o For each node in copiaNos:

{jt.sep.potentialTable, jt.clique.potentialTable}[node.associatedClique]

▪ If it's a ProbabilisticNode: look for the separator that contains this node which has the smallest potentialTable and set this separator as associatedClique of the node.

▪ If it's a DecisionNode or it doesn't exist a separator which contains this node: look for the clique that contains this node which has the smallest potentialTable and set this clique as associatedClique of the node.

o For each node in copiaNos:

▪ Init marginalList as a new array of Floats. [node.marginalList]

        - Set marginalList values to values obtained from nodo.cliqueAssociado.potentialTable.
          {node.cliqueAssociado.potentialTable} [node.marginalList]

From JUNCTION_TREE_COMPILED_STATE to LINKAGE_TREE_MADE_STATE

For each linkage:

- Clear the linkList. [linkage.linkList]
- Assign a new jt (junction tree) to this linkage. [linkage.jt]
- Call makeCliqueList(n1.jt.clique0) method, where makeCliqueList(Clique c) does the following:
  {n1.jt.cliques}
    - Create a new clique (b) with the nodes intersection between this linkage and c.
    - Add b to jt.cliques. [linkage.jt.clique]
    - Add to jt.linkList a new link with b as host0. [linkage.jt.linkList]
    - For each clique child of c:

      [linkage.jt.clique, linkage.jt.linkList, linkage.jt.clique.parent, linkage.jt.clique.child]
        - Call makeCliqueList(child) obtaining b2, the new intersection clique created from the child.
        - Set b as parent of the child.
        - Add b2 as child of b.
    - Return b.
- Call remove1stPass method where, for each link in linkage.jt.linkList, the following is done:
  [linkage.jt.linkList]
    - If the clique hasn't children:
        - Remove nodes from linkage.jt.linkList.link.clique that are already in the parent of this clique.
        - If the clique hasn't nodes, remove this link.
- Call remove2ndPass method where, for each link in linkage.jt.linkList, the following is done:
  [linkage.jt.linkList]
    - If all nodes in link.clique are already in the parent: remove link.
    - If any of the children of the parent of this clique (a brother) has all of its nodes: remove link.
- Call initTables method, where the following is done:

[jt.separators, linkList.link.clique.potentialTable, jt.separators.separator.potentialTable]

- InitSeparators: for each linkage.jt.clique:
  - For each clique.child:
    - Construct a new separator with the nodes of the intersection of its nodes.
- For each linkList.link.clique.potentialTable:
  - Add as many variables as nodes in the clique.
  - Set all of these variables to 1.
- For each jt.separators.separator.potentialTable:
  - Add as many variables as nodes in the separator.
  - Set all of these variables to 1.

From LINKAGE_TREE_MADE_STATE to BELIEFS_INITIALIZED_STATE

Each subnetwork:

- Collect beliefs by doing the following:
  - Wait for receiving a belief update from each adjacent subnetwork and then, send a belief update to this subnetwork's parent if it exists. To send a belief update to this subnetwork's parent, send the information of the potentialTable of the sharing clique (equivalent to *host1* of a link) removing the information corresponding to private nodes.
    sendTransferBeliefsToParent
- Distribute beliefs by doing the following:
  - Wait for receiving a belief update from this subnetwork's parent (if it exists) and then, send a belief update to each adjacent subnetwork.
    sendTransferBeliefsToAdjacent

updateBeliefs(netToUpdate, fromNet):

- For each links.linkage:
  - Linkage.absorb(fromAdjacent = true)
    - For each linkage.linkList.link:
      - Link.absorbIn(fromAdjacent)
        [link.originalLinkTable, link.newLinkTable, link.clique.potentialTable]
    - RemoveRedundancy:
      - For each linkage.jt.separator:
        - Remove from separator.probabilityFunction all variables which are in linkage.clique2 and are not in separator.
        - Copy to the separator.probabilityFunction values in linkage.clique2.

        - For each linkList.link which clique is linkage.clique2: link.removeRedundancy()
    - For each linkage.linkList.link:
        - Link.absorbOut(fromAdjacent) [link.originalLinkTable, link.newLinkTable, link.v0/v1.potentialTable]
    - For the subnetwork to what beliefs propagate:
        - Absorb2()

Link.absorbIn(fromAdjacent):

- Copy clique.potentialTable to originalLinkTable {link.clique.potentialTable}[link.originalLinkTable]
- Use the information received as newLinkTable.

- Copy values in newLinkTable to link.clique.PotentialTable. {link.newLinkTable}[link.clique.potentialTable]

In short, take the corresponding part of clique link.v1/v2 and set it as link.clique.potentialTable.

Link.absorbOut(fromAdjacent):

- Divide link.newLinkTable by originalLinkTable. {link.originalLinkTable}[link.newLinkTable]
  Multiply link.v0/v1.potentialTable by link.newLinkTable.potentialTable obtained. {link.newLinkTable}[link.v0/v1.potentialTable]

SubNetwork.Absorb2():

This.jt.consistency()

This.updateMarginals() [nodeList.node.marginalList]

From BELIEFS_INITIALIZED_STATE to COMPILATION_DONE_STATE

Perform updating of beliefs according to list of received beliefs.

Every time a new belief update is received, the subnetwork goes back to this compilation state and performs an update of the beliefs with the source subnetwork.

IV - 2.2.b.3. Absorbing pending compilation states

Between each compilation state, *absorbPendingCompilationState* method is called.

It reads the list of *pendingCompilationStates* and change the *currentCompilationState* according to the petitions received.

This way, we can go back in the compilation process depending on the requirements introduced in this list.

For example, let BELIEFS_INITIALIZED_STATE pending compilation state be added to the list, when this pending compilation state is absorbed, the compilation goes back to BELIEFS_INITIALIZED_STATE, where beliefs have already been initialized. Then, the compilation continues from that compilation state.

IV - 2.2.b.4. Adding and propagating Beliefs

Add Finding

- Add the new evidence to the list of pending Beliefs and add a new pending compilation state BELIEFS_INITIALIZED_STATE to go back to this state if necessary.
- When compilation reaches BELIEFS_INITIALIZED_STATE, the list of pending Beliefs is read and the following is done:
    - Set the evidence of this node to the number passed. [node.evidence]
    - Set the marginal of this node according to the evidence set. {node.evidence} [node.marginalList]

Update Evidences

- When a new finding is added, a new belief update is sent to each adjacent subnetwork.
- If a subnetwork receive a new belief update, it propagates this to each adjacent, but to the subnetwork from which this belief update has been received. If receive two belief updates at the same time from different subnetworks, it propagates this belief update to all its adjacent subnetworks, included those from which it has received these belief updates.

IV - 2.2.b.5. Attending received messages

Communication framework used, JGroups (see II - 6.1), allows asynchronous reception of messages. This means that the reception of the messages is executed in a different thread than compilation or other processes. Thus, a message can be needed to be received at any moment, although its use be done at a different moment.

That is why there are several queues for different purpose received messages. Each message is labeled with one of the following tags. In response to each received message, a way of procedure is shown below.

publishNodes

- Add the information received about a subnetwork in the *snInfo* list.
- If any information is added, add a new pending compilation state INITIAL_RESET_STATE.

verifyCycles

- Try to mark all that nodes which can be marked. When a public node is marked, send a new *distributeMark* message.
- If all nodes have been marked, send a *notifyCycleVerificationDone* to the root.
- If is the root subnetwork and all other subnetworks have completed the verification of cycles, send a *notifyCycleVerificationDone* to all subnetworks.

distributeMark

- Set the node passed as marked. Another subnetwork has marked it.

notifyMarkedChildNodes

- Set the child nodes from the sender subnetwork as marked to allow the marking of this node.

notifyCycleVerificationDone

This message has a different meaning depending on which subnetwork sends it.

- If the sender is a subnetwork different from root, this message means that all nodes of this subnetwork have been marked successfully.
- If the sender is the root subnetwork, this message means that all subnetworks have marked all its nodes successfully.

addMarkovArcsFromMoralization or addMarkovArcsFromTriangulation

- Add the markov arcs passed to the list of received markov arcs corresponding to the sender subnetwork.
- Add a new pending HYPERTREE_DONE_STATE compilation state.

transferBeliefsToAdjacent or transferBeliefsToParent

- Add a new pending belief update with the data received.

IV - 2.2.c. The keys of this architecture

After the full-detailed description of this architecture presented in section IV - 2.2.b.2, an overview of the more important characteristics of this architecture is shown in this section. These all features have been adopted during the analysis of the solution proposed and form the base of the developed architecture.

IV - 2.2.c.1. Reactive behavior

In this architecture, each node has a reactive behavior. This means that each Hypernode do not waste time of process trying to discover other Hypernodes or information about them.

To know the existence of other Hypernodes, each Hypernode that joins the group in JGroups (that is initialize the communication process) publish the information that all other Hypernode need to know about it. Publish the identifiers of all its public nodes, the identifier of the Hypernode and the markov arcs that this Hypernode can share with other Hypernodes. This first message is sent to all Hypernodes in the MSBN and allows all Hypernodes to know its place in the Hypertree.

Likewise, a Hypernode do not need to ask for the fill-ins to its adjacent Hypernodes. Instead, when a new fill-in is created, it is immediately communicated to the adjacent Hypernodes, which have to recompile to incorporate the new information received.

Similarly, when new information is received from other Hypernode, according to the type of this information, a Compilation Pending is added to the corresponding queue. Thus, the Hypernode can recompile only those necessary steps to incorporate that information.

When a belief update is received from another subnetwork, this is queued to be absorb when convenient.

IV - 2.2.c.2. Compilation executed in a separated thread

An only Hypernode has several threads to perform different task during its work. Mainly there exist three threads associated to a Hypernode. The first thread exists only during the loading of the Hypernode and is the responsible for the creation of the Hypernode and the other threads without affect to the thread that launches this Hypernode. The second thread is created when JGroups component is created associated to the Hypernode, and is the responsible for receiving messages and adding the received information to the corresponding queue to be processed later. The last thread is the compiler thread. Compiler thread is always trying to reach the highest state of compilation. When the whole compilation is performed, this thread sleeps until a new event occurs or any new information is received and going back in compilation is needed.

Summarizing, new events are always queued to be read when convenient.

IV - 2.2.c.3. Distributed Verification of cycles

The greatest difficulty in the developed system is the dynamism that it can manage. Before a Hypernode can have reached compilation, a new Hypernode may have been added to the MSBN.

In this scenario, we need to be able to check that the state of the created MSBN is the correct. That means we need to check that there is not any cycle in the BN formed by the MSBN.

To achieve this, a technique about marking nodes is followed, as described in [23]. Over the technique described in this article, we have included several modifications that are summarized below.

To carry out the verification, messages need to be sent between adjacent Hypernodes. Those messages could be confused between different Hypertrees, causing a wrong verification result. To avoid this, when a new Hypertree is built, after the discovering or falling of a Hypernode, root Hypernode send a message to all Hypernodes assigning a new verification identifier. All messages sent during this verification have this assigned identifier and verification messages that contain an identifier different that current are dropped. Finally, when a Hypernode verifies that it does not contain any cycle, this information is sent to the root, which is the responsible for the knowledge of the state of the verification of the cycles in the MSBN.

To mark a node, sometimes a Hypernode needs to know if all Hypernodes that contains any children of this node have already marked it. To allow this, when a Hypernode mark all children of a public node, send a message to inform all Hypernodes that share this node about the event. As each Hypernode knows, thank to the information sent in the publication of nodes made at the beginning, the identifiers of all nodes relative to shared nodes, it can be achieved easily.

IV - 2.2.c.4. Moralization with all fill-ins received in the history

Fill-ins mean the same whether they come from moralization or from triangulation. Thus, when a new fill-in is received in a Hypernode, compilation return to the beginning of moralization. Across the history of a Hypernode, all received fill-ins are saved and included during moralization.

However, when a Hypernode falls, all its adjacent Hypernodes check for the fill-ins that the fallen Hypernode had sent to them, and remove those fill-ins.

As after the adding of a fill-in moralization results could have been changed, it is necessary that triangulation be repeated again to be consistent with the results.

IV - 2.2.c.5. Linkage belongs to parent subnetwork

In the single-agent architecture, Linkages (the shared part between two Hypernodes) do not belong to any Hypernode. They are just an object that the manager owns. However, this is an obstacle to carry out a well-distributed system that can be scalable.

In this architecture, Linkages belong to the Hypernode that is associated by that Linkage and has a highest position in Hypertree. This way, from two Hypernodes connected by a Linkage, the parent one does not have any problem to carry out operations over the Linkage. Nevertheless, the child Hypernode cannot operate directly over the Linkage. To solve this issue, child Hypernode have to perform those operations to what it needs its private information before sending the changes to perform in the Linkage to the parent Hypernode. Then, when the parent Hypernode receives these semi-performed changes, it finishes the operation applying them over the Linkage.

To maintain privacy, Linkages cannot have whole cliques that are connected by them. Then, its Links only have *host0* and *clique*. The *host1* clique is not explicitly saved. The child subnetwork adds the information received to the proper table by knowing only the names of the public nodes shared in the clique.

IV - 2.2.c.6. Hypertree recalculated in each Hypernode

To avoid centralized paradigm, each Hypernode has to calculate the place it occupies in the Hypertree. As all Hypernode have all relevant information to perform this operation, the same operation is replicated in each Hypernode. This way, no messages are needed to build a correct Hypertree, and segmentation of a MSBN in several ones is allowed if the Hypernodes that connect them fail. It can be said that each Hypernode is independent from others, achieving the autonomy in each part of the MSBN.

IV - 2.2.c.7. Essential information distributed to all nodes

Each Hypernode only shares the public information about itself. That allows the maintaining of privacy, which is one of the main advantages of MSBN. Thus, each Hypernode only retain the public information about other Hypernodes. This feature can make certain operations more complicated than what they are sharing all information.

IV - 2.2.d. The issues of this architecture

Although developed architecture have fulfilled our expectations, there are some problems or not desired featured that need to be taken into account. These features can be the subject of future work to avoid potential problems that may be arisen.

IV - 2.2.d.1. Infinite updating cycle

In some situations, the including of new evidences can be done in different Hypernodes at the same time. As the system is distributed and each Hypernode works in parallel, depending on the situation and the delivery of the corresponding messages, updating cycle could result into an infinite cycle of beliefs updating that have no end. This problem is fully discussed in [24]. This is not properly a problem, in fact it has never been observed during the development of this project, but could result into undesired behaviors.

V - Test Plan

Test plan has the task of verifying the proper functioning of the developed system by checking that it meets the requirements of performance.

To facilitate the understanding of the test plan used in the project, this chapter is structured by presenting, first, the description of the model that has been used for test cases and, secondly, the description of such cases and test results.

V - 1. Test Specification

In this section, we present an overview of the tests that have been done to verify the correction and well functioning of the developed system.

Since synchronous architecture is not the target of this project, all tests described in this section are related to Iterative Architecture, which should be the center of the attention of the reader and is the great result of our work.

First, we present the test plan that has been done on the system and, second, an analysis of requirements to check if they have been achieved and how much has been done.

V - 1.1. Unit Tests

Unit tests check that concrete parts of our developed system work correctly. As the developed builds on already developed frameworks, such as UnBBayes or JGroups, we assume that those frameworks work properly. Thus, unit tests corresponding to how an only Hypernode must react under changes or events need to be implemented.

There exist two type of tests, those that check that something is sent correctly (Sender Tests), and those that check that the reaction showed when something is received is performed correctly (Receiver Tests).

V - 1.1.a. Sender Tests

V - 1.1.a.1. Initial Information – Sender Test

This test checks that a Hypernode initializes its compilation properly sending the corresponding publication of shared information. In addition, it checks that the information contained in that message is complete and correct.

V - 1.1.a.2. Sharing fill-ins from local moralization – Sender Test

This test checks that a Hypernode sends correctly all fill-ins that have been added during local moralization.

V - 1.1.a.3. Sharing fill-ins from triangulation – Sender Test

This test checks that a Hypernode sends correctly all fill-ins that have been added during triangulation process.

V - 1.1.a.4. Cycle verification – Sender Test

This test checks that a Hypernode send correctly the information needed for the verification of non-existence of cycles in the whole MSBN. In addition, it checks that its own verification is performed properly when communications are not needed.

V - 1.1.a.5. Finalization of cycle verification – Sender Test

This test checks that a Hypernode send the corresponding message to communicate to root node that it has finished its own verification of cycles.

V - 1.1.a.6. Transferring beliefs to parent – Sender Test

This test checks that a Hypernode, which is child of other Hypernode, sends the proper message to transfer its beliefs to its parent. It also checks that all data in the probability tables passed is correct.

V - 1.1.a.7. Transferring beliefs to adjacent subnetwork – Sender Test

This test checks that a Hypernode, which is parent of other Hypernode, sends the proper message to transfer its beliefs to its adjacent Hypernode. Additionally, it checks that the proper operations over Linkage are performed correctly by checking the results obtained. It also checks that all data in the probability tables passed is correct.

V - 1.1.a.8. Stopping a Hypernode – Sender Test

This test checks that a Hypernode is stopped correctly. It is very important, among other reasons, to allow an efficient test plan. Many problems have been found due to existing Hypernodes from past tests that continue its communication in the network.

V - 1.1.b. Receiver Tests

V - 1.1.b.1. Initial Information – Receiver Test

This test checks that a Hypernode properly incorporates the received initial information about a new Hypernode in the group. In addition, it checks that the information incorporated to the knowledge of the Hypernode is complete and correct.

V - 1.1.b.2. State Information – Receiver Test

This test checks that a Hypernode, which is new in the group, receives an incorporates correctly the information about existing Hypernodes, which is passed as the state of the group.

V - 1.1.b.3. Sharing fill-ins from local moralization – Receiver Test

This test checks that a Hypernode receives and incorporates correctly all fill-ins that have been added during local moralization of other Hypernode. In addition, it must result in the return to a passed compilation state.

V - 1.1.b.4. Sharing fill-ins from triangulation – Receiver Test

This test checks that a Hypernode receives and incorporates correctly all fill-ins that have been added during triangulation process of other Hypernode. In addition, it must result in the return to a passed compilation state.

V - 1.1.b.5. Cycle verification – Receiver Test

This test checks that a Hypernode receives correctly the information needed for the verification of non-existence of cycles in the whole MSBN and continue its own verification with the new information added. In addition, it checks that its own verification is performed properly with the new information.

V - 1.1.b.6. Finalization of cycle verification – Receiver Test

This test checks that the root Hypernode receives the corresponding messages to communicate that other Hypernodes have finished its own verification of cycles. In consequence, it checks that root Hypernode checks if all Hypernodes have already finished and if the message to communicate that cycle verification has finalized is sent correctly.

V - 1.1.b.7. Transferring beliefs to parent – Receiver Test

This test checks that a Hypernode, which is parent of other Hypernode, receives the proper message to receive new beliefs from its child. It also checks that all data in the probability tables passed is correct, and that all beliefs and marginals are correct after performing the beliefs update.

V - 1.1.b.8. Transferring beliefs to adjacent subnetwork – Receiver Test

This test checks that a Hypernode, which is child of other Hypernode, receives the proper message to receive new beliefs from its parent. It also checks that all data in the probability tables passed is correct, and that all beliefs and marginals are correct after performing the beliefs update.

V - 1.2. Integration Tests

Once Unit test have been passed, the complete functionality of a Hypernode in the MSBN is probed. Nevertheless, when more than two Hypernodes form part of the MSBN, different situations are presented that need to be taken into account.

For this purpose, different MSBN have been developed to allow the performing of those tests. For each proposed MSBN, full compilation process is checked and, after compilation has finished, some evidences are added to check that belief updating is performed correctly.

The tests contained in this section are sorted according to the size of the MSBN used. Thus, the first test presented correspond to a little MSBN that consists of just two Hypernodes, each of which has two nodes, and the last test presented correspond to a huge MSBN that consist of five Hypernodes, each of which have at least four nodes. In all cases, used MSBNs are extremely small compared with those that would be used in a real scenario.

V - 1.2.a. Test using Ridiculous MSBN

This test uses the smallest MSBN that can be built. The MSBN used in this test is shown in Figure V-1.

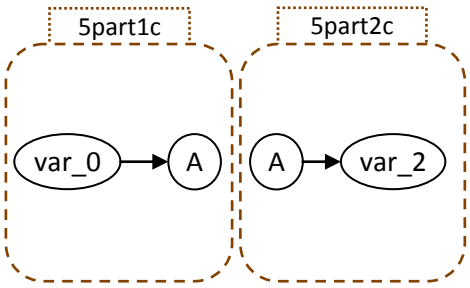


Figure V-1 Ridiculous MSBN representation

V - 1.2.b. Test using Little MSBN

This test uses a small MSBN that contains a V structure. The MSBN used in this test is shown in Figure V-2.

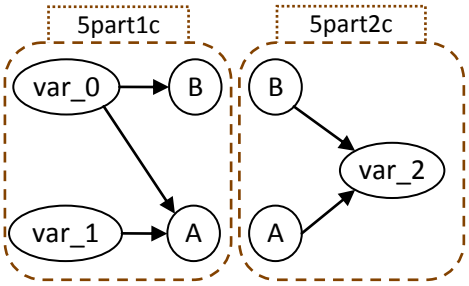


Figure V-2 Little MSBN Representation

V - 1.2.c. Test using Medium Size MSBN

This test uses a three-Hypernode MSBN that involves the adding of fill-ins during moralization process but not during triangulation. The MSBN used in this test is shown in Figure V-3.

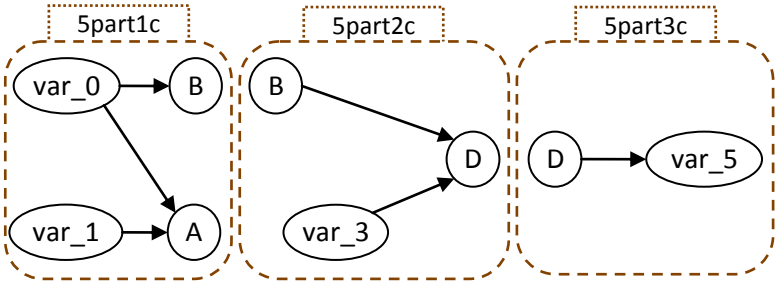


Figure V-3 Medium size MSBN representation

V - 1.2.d. Test using 2-subnetwork MSBN

This test uses a simple two-Hypernode MSBN that allows the experimentation of testing with a non very difficult MSBN. The MSBN used in this test is shown in Figure V-4.

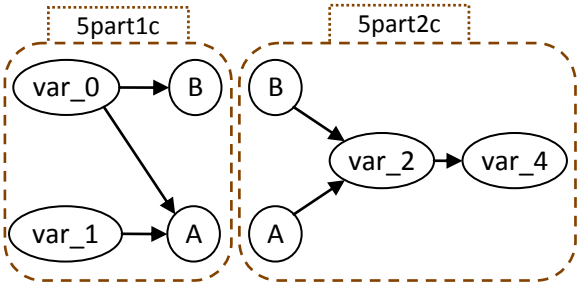


Figure V-4 2-subnetwork MSBN representation

V - 1.2.e. Test using 3-subnetwork MSBN

This test uses a three-Hypernode MSBN with a not quite difficult structure that involves the adding of fill-ins during moralization and triangulation processes. The MSBN used in this test is shown in Figure V-5.

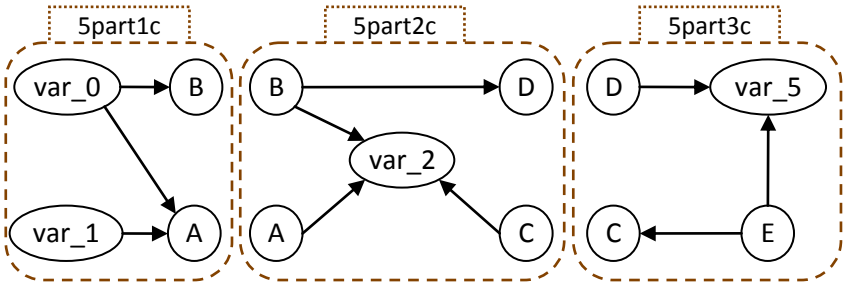


Figure V-5 3-subnetwork MSBN representation

V - 1.2.f. Test using Acyclic MSBN

This test uses a three-Hypernode MSBN that contains a cycle among the three Hypernodes to check that cycle verification works properly. The MSBN used in this test is shown in Figure V-6.

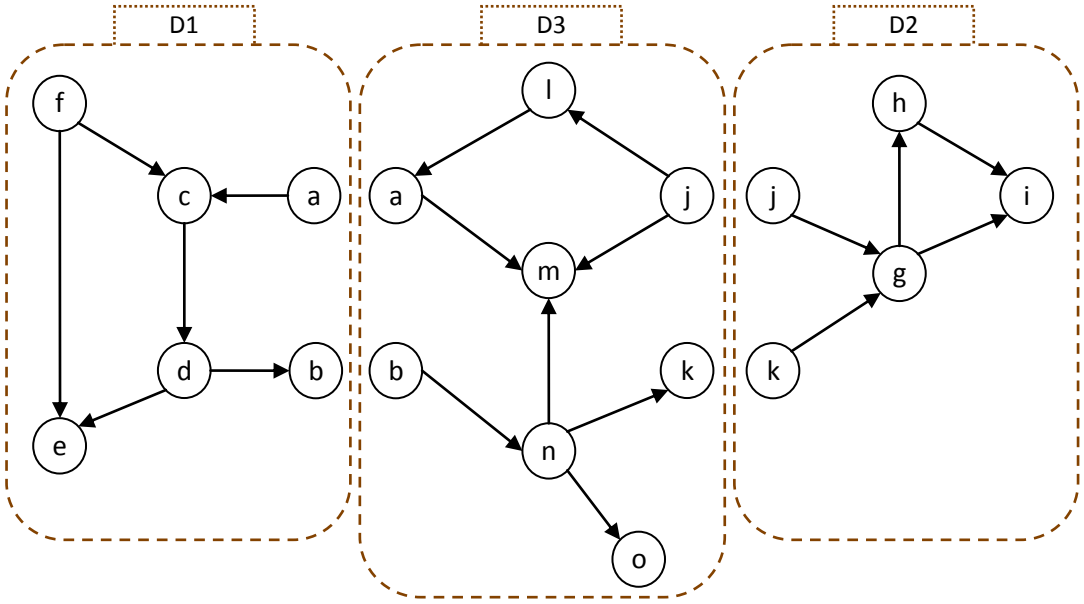


Figure V-6 Acyclic MSBN representation

V - 1.2.g. Test using Cyclic MSBN

This test uses a three-Hypernode MSBN that does not contain any cycle to check that cycle verification works properly. The MSBN used in this test is shown in Figure V-7.

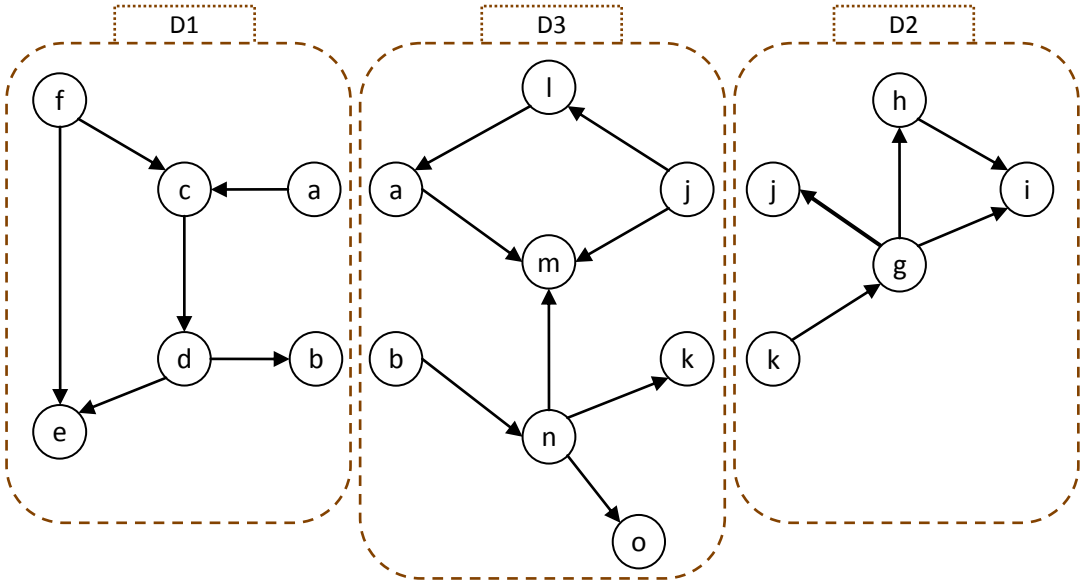


Figure V-7 Cyclic MSBN representation

V - 1.2.h. Test using 5partc MSBN

This test uses a commonly used MSBN in MSBN framework developing. This MSBN contains several V structures that provoke the adding of fill-ins in moralization, as well as during triangulation.

Moreover, this MSBN has a complex Hypertree structure building due to shared nodes between Hypernodes is a little intricate. Such structure can result into Hypertree cycles if is not built correctly, fact by which this is a very suitable MSBN to prove that our framework is working properly.

The MSBN used in this test is shown in Figure V-8.

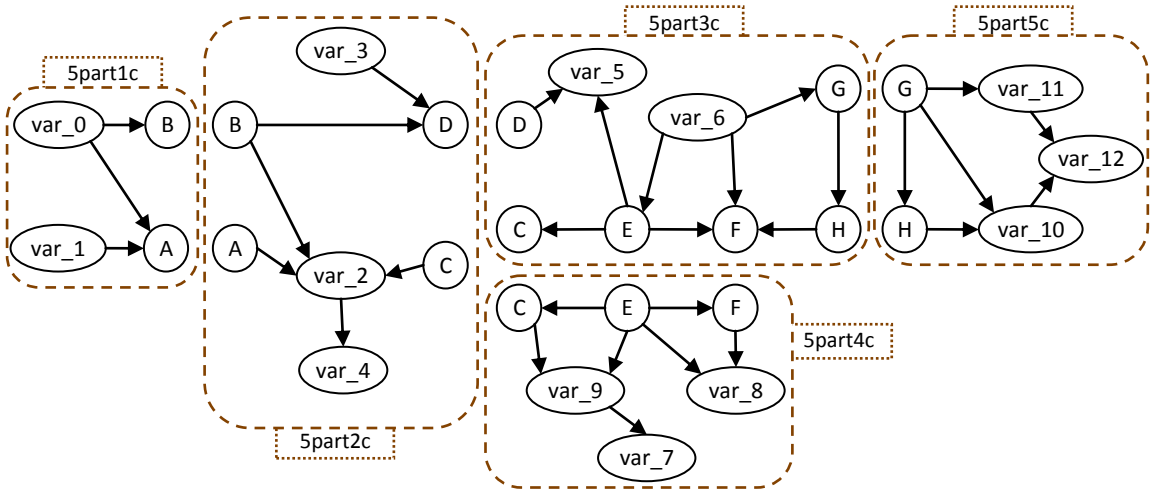


Figure V-8 5partc MSBN representation

V - 1.3. Adaptability Tests

To check that adaptability requirements are satisfied, the following adaptability tests are done.

V - 1.3.a. Apparition of a new Hypernode

This test uses a three-Hypernode MSBN known as 3-subnetwork MSBN (see Figure V-5). The test starts with an only Hypernode alive. Then, two more are created and they should be added to the existing MSBN.

V - 1.3.b. Falling of an existing Hypernode

This test uses a three-Hypernode MSBN known as 3-subnetwork MSBN (see Figure V-5). The test starts with the three Hypernodes alive and calls stop method for the second one. Then, when the MSBN is restructured, the first Hypernode is stopped and the MSBN needs to be restructured again.

V - 1.3.c. Two existing MSBN join into an only MSBN

This test uses a three-Hypernode MSBN known as 3-subnetwork MSBN (see Figure V-5). The test starts with two non-adjacent Hypernodes alive, which are considered as two separated MSBNs. Then, the third Hypernode is created, what should result into the restructuration of the whole MSBN to be an only MSBN.

V - 2. Test Results

In this section, we analyze the results obtained to the tests described in section V - 1. To facilitate the task of revise these results, we have decided to show them in

Table V-1.

Each test has an identifier that is taken from the numeration of this report. Thus, test from point V - 1.1.b.3, will be associated with the identifier 1.b.3.

For each test, is associated a punctuation that can vary from zero to five, meaning zero that this test is not passed at all, and five that this test satisfy all requirements presented. Punctuations that are in the middle mean, one that this test satisfy some requirements, two that this test satisfy hardly all requirements, and four that this test satisfy all requirements but can present some problems depending on the scenario conditions.

Id.	Punctuation	Comments
1.a.1	5	✓
1.a.2	5	✓
1.a.3	5	✓
1.a.4	5	✓
1.a.5	5	✓
1.a.6	5	✓

1.a.7	5	✔
1.a.8	4	When a test fails or need to be stopped manually due to it is blocked, stop is not performed correctly. In that situation, its needed to execute manually `killall -9 java` (in linux os) to eliminate all existing processes that continue running.
1.b.1	5	✔
1.b.2	5	✔
1.b.3	5	✔
1.b.4	5	✔
1.b.5	5	✔
1.b.6	3	Although verification is performed perfectly, the information resulting from this process is not used yet. It should be necessary a checking of the structure verification state before the beginning of belief updating.
1.b.7	4	Iterative paradigm involves a certain degree of uncertainty about the time in which the beliefs are updated. Thus, depending on the communication situation, verification of this test can incorrectly fail due to those time issues. Although several mechanisms have been developed to avoid these situations, those mechanisms are not always working correctly, what implies some incorrect situations.
1.b.8	4	Iterative paradigm involves a certain degree of uncertainty about the time in which the beliefs are updated. Thus, depending on the communication situation, verification of this test can incorrectly fail due to those time issues. Although several mechanisms have been developed to avoid these situations, those mechanisms are not always working correctly, what implies some incorrect situations.
2.a	5	✔
2.b	5	✔
2.c	5	✔
2.d	5	✔
2.e	4	It hardly never fails due to JGroups merging of views or dropped or disordered of messages.
2.f	5	✔
2.g	5	✔
2.h	4	It almost always fails due to JGroups merging of views or dropped or disordered of messages.
3.a	5	✔
3.b	5	✔
3.c	5	✔

Table V-1 Test results summary

VI - Case study: FTTH

The case study proposed builds on FTTH scenario.

In this chapter, we present an architecture that uses the developed MSBN framework to perform the diagnosis and solving of failures in the FTTH scenario.

First, a brief introduction to FTTH scenario is given. Second, a description of the proposed reasoning system is given. Third, model proposed to simulate FTTH scenario is presented. Finally, the corresponding MSBN that allows the diagnosis, inference and reasoning over the proposed simulated scenario is described.

VI - 1. FTTH Scenario

Our case study is the scenario described in this section. This scenario is based in the FTTH-GPON (Fiber to the Home - Gigabit Passive Optical Network) architecture. This architecture use passive splitters to carry the signal to the final users. In this architecture, every user receives all information of the tree, but encrypting, each user processes only his information.

In this scenario, there are the following important devices:

- OLT – Optical Line Terminal (Active device)
- Splitter (Passive device)
- ONT – Optical Network Terminal (Active device)
- RGW – Router Gateway (Active device)

This list of devices exists per user in the scenario. In other words, there is at least one of these components per user. Several devices (like OLT or splitters) are shared to offer services to several users at the same time.

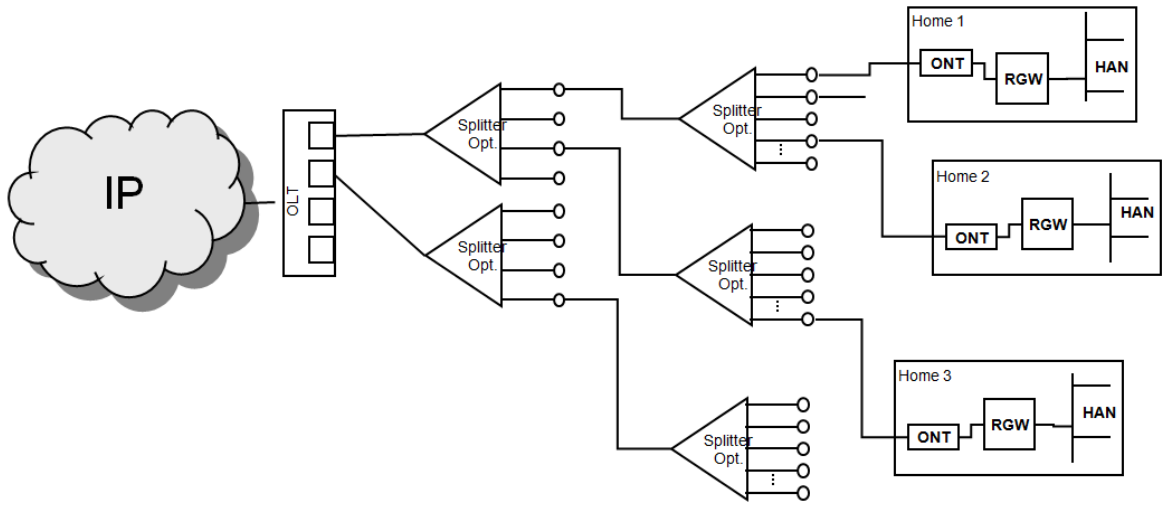


Figure VI-1 FTTH scenario

With the capacity of this technology, it is able to offer 2.5 GB/s to 64 users. In other words, each line from OLT has this capacity, but this line is shared between up to 64 users. Despite sharing the fiber, this architecture supports the providing of several services (IPTV, VoIP, data connection, etc.) at same time. However, even more, this technology can be improved with WDM (Wavelength-Division Multiplexing) to obtain more bandwidth and, at this way, provide more services.

This scenario consists on an operator that provides several services to final users through a FTTH-GPON architecture based in the image shown below.

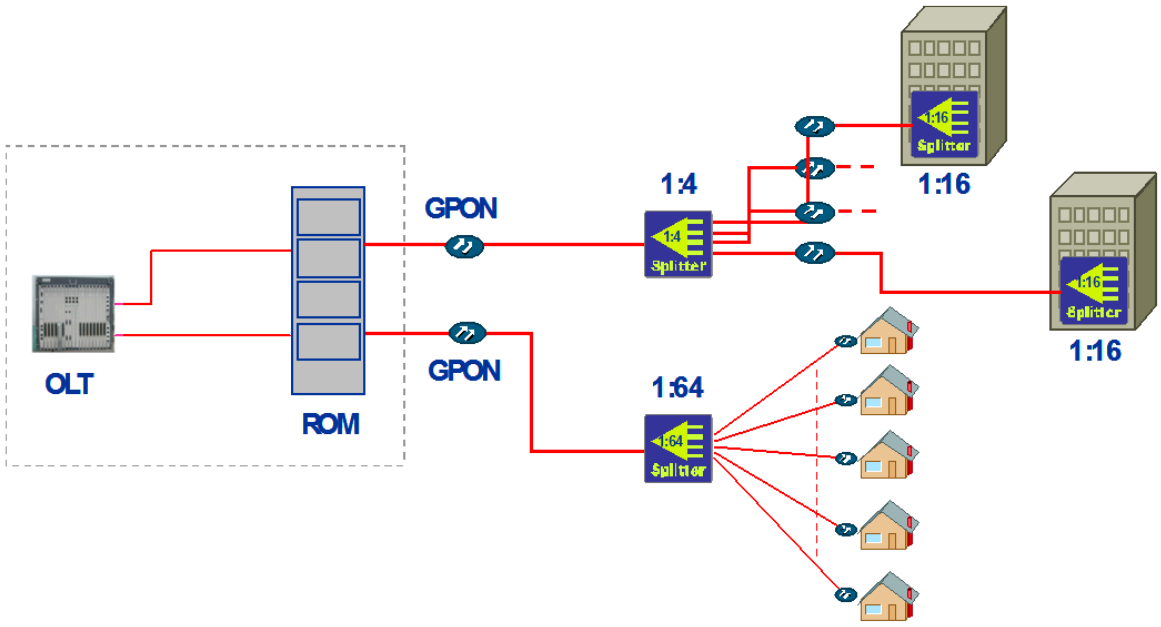


Figure VI-2 FTTH scenario with splitter divisions

The topology of the access network is shown in the figure: two splitters between each ONT and its OLT. The first splitter from OLT divides the signal into 4, and the second one divides it into 16. Then, 64 users (one ONT and one RGW per user) per OLT have access to the network.

VI - 2. Proposed reasoning system

The reasoning system proposed over the FTTH scenario consists of a Multi-agent system distributed across the whole scenario.

Each active device has an associated agent, which perform tests over the device and extracts some information to allow reasoning. Each of those agents has an associated Hypernode that belongs to a MSBN shared between all agents. According to the collected information about the device, the Hypernode related to the agent is updated, allowing global inference and reasoning having into account the information collected in all devices.

To allow this system, each active device in the FTTH scenario should have a higher processing capacity than in a normal FTTH scenario.

Exploiting the feature of the developed MSBN framework that make it adaptable to falls and recovering of nodes, this whole system is able to reason in spite of the situation of the scenario. This is, if an optic fiber is damaged, the reasoning system can continue its work and discover where the failure is, despite of it has unconnected nodes.

Internally, each agent can have different structures. Nevertheless, in our work we have chosen to use BDI agents. That mean each agent is divided internally in a Beliefs-Desires-Intentions structure.

Beliefs are the knowledge that the agent contains. Beliefs represent the informational state of the agent, in other words its beliefs about the world (including itself and other agents). Beliefs can also include inference rules, allowing forward chaining to lead to new beliefs. Using the term

belief rather than knowledge recognizes that what an agent believes may not necessarily be true (and in fact may change in the future).

Desires represent the motivational state of the agent. They represent objectives or situations that the agent would like to accomplish or bring about.

Intentions represent the deliberative state of the agent, what the agent has chosen to do. Intentions are desires to which the agent has to some extent committed. In implemented systems, this means the agent has begun executing a plan.

Inside BDI model, the MSBN is another belief contained in the belief base of the agent. Thus, MSBN information can be accessed as one more piece of agent's knowledge and is used to control the behavior of the agents.

All agents can reason together using MSBN framework. This framework abstract programmer from belief updating and makes easier the way of reasoning in the BDI agent environment.

VI - 3. Proposed simulation model

To simulate the reasoning system proposed in section VI - 2 a simple model is proposed. MASON simulation framework provides an easy way of simulate an agent environment with little effort [25].

To simulate the FTTH scenario, a class hierarchy is needed. In this set of classes, each device has a software representation with different available states that must be related with real ones.

For this purpose, classes needed to represent FTTH scenario have been developed. To facilitate the comprehension of the reader, class diagram shown in Figure VI-3 is provided.

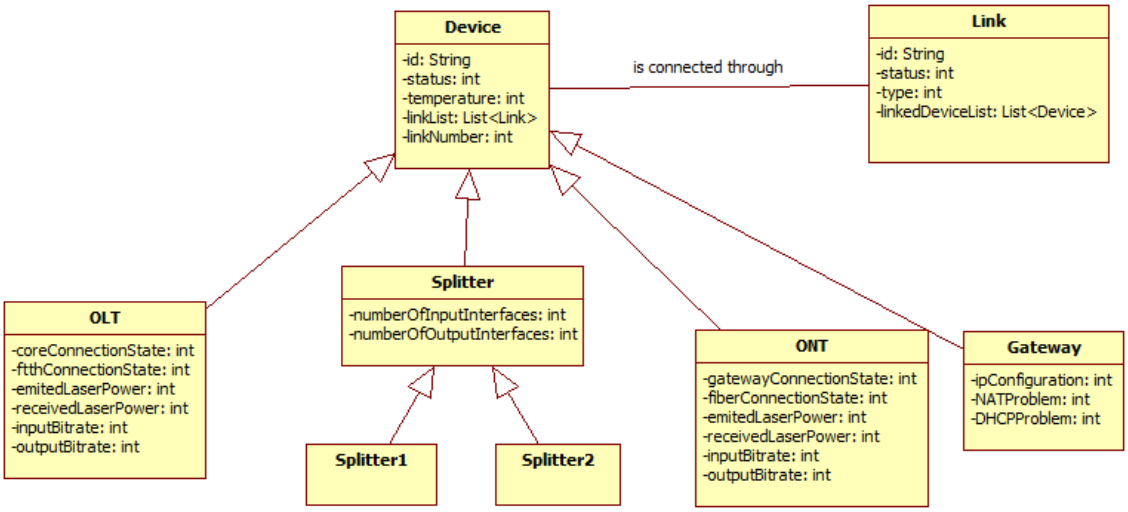


Figure VI-3 Class diagram for the proposed simulation model

According to FTTH scenario, the proposed simulation model have a representation for one of each devices included in the scenario. The more important device is OLT, which has several interfaces and is connected to the core of the network. Thus, several links can connect the OLT to the first splitter (splitter1). This first splitter splits the communication channel into four fibers to the second splitter (splitter2). The second splitter splits the communication channel into 16

fibers to the OLT. The OLT is the responsible for converting optic signals to electric signals. Therefore, if we abstract from the optic fibers and splitters, which are all passive devices, OLT and ONT devices have a direct communication. Following the ONT, Gateway is connected to provide connection to the HAN (Home Area Network). Depending on the situation, different users are connected to the HAN.

VI - 4. MSBN for the case study

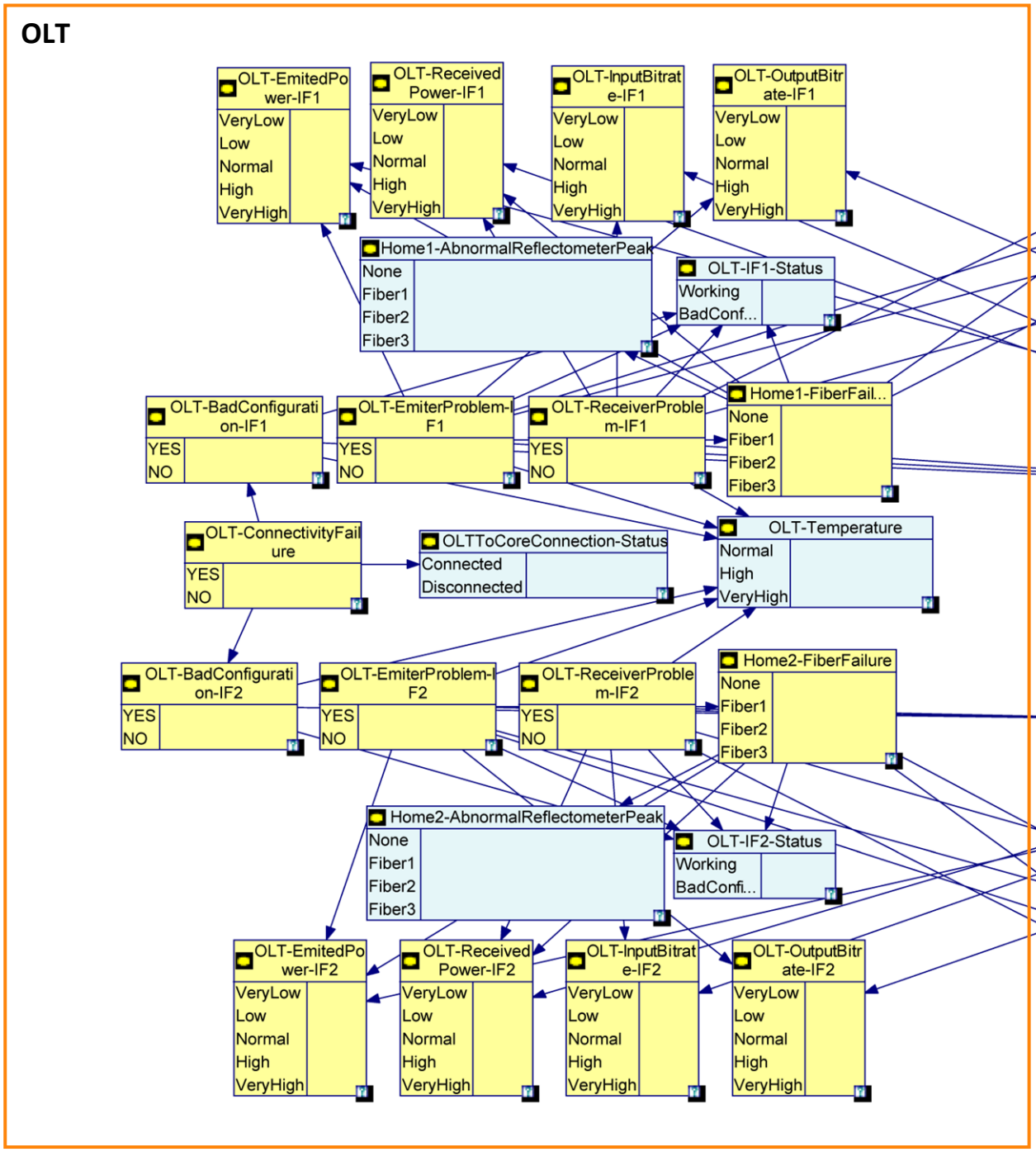
The proposed case study deals with a streaming connection between a streaming server and a client. Both server and client are situated in a different HANs of the FTTH scenario.

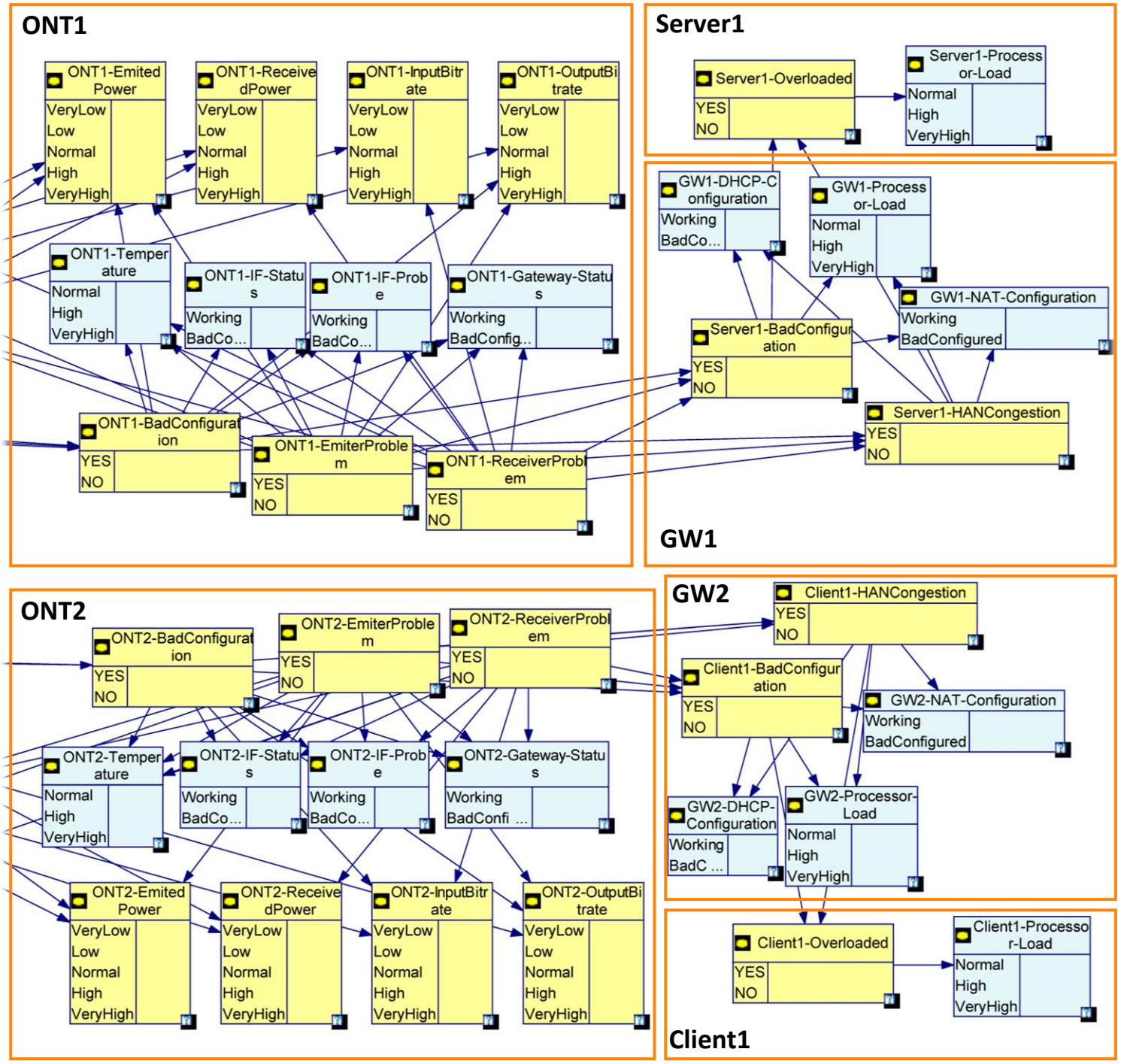
Distributed across the whole scenario there is a MSBN as can be seen in Figure VI-4.

It is divided according to devices present in the scenario, and only information in yellow is shared between Hypernodes.

In general, evidences taken from the scenario are colored in blue. However, several shared nodes are evidences too, such as OLT-ReceivedPower or ONT1-InputBitrate.

Figure VI-4 MSBN for FTTH Scenario




ONT1
ONT1-Emited Power
VeryLow
Low
Normal
High
VeryHigh
ONT1-ReceivedPower
ONT1-InputBitrate
ONT1-OutputBitrate
ONT1-Temperature
ONT1-IF-Status
Working
BadCo...
ONT1-IF-Probe
ONT1-Gateway-Status
BadConfig...
ONT1-BadConfiguration
YES
NO
ONT1-EmiterProblem
ONT1-ReceiverProblem
Server1
Server1-Overloaded
Server1-Processor-Load
GW1-DHCP-Configuration
GW1-Processor-Load
GW1-NAT-Configuration
BadConfigured
Server1-BadConfiguration
Server1-HANCongestion
GW1
ONT2
ONT2-BadConfiguration
ONT2-EmiterProblem
ONT2-ReceiverProblem
ONT2-Temperature
ONT2-IF-Status
ONT2-IF-Probe
ONT2-Gateway-Status
BadConfi ...
ONT2-Emited Power
ONT2-ReceivedPower
ONT2-InputBitrate
ONT2-OutputBitrate
GW2
Client1-HANCongestion
Client1-BadConfiguration
GW2-NAT-Configuration
GW2-DHCP-Configuration
BadC ...
GW2-Processor-Load
Client1-Overloaded
Client1-Processor-Load
Client1

Conclusion

Most tasks performed by humans, as far as communications systems are concerned, could be performed by expert systems. Specifically, referring to network issues, more dynamic and precise systems and offering better features could be developed aided by expert systems. The future of our networks is self-organizing networks: self-configuration, self-optimization and self-healing. Expert systems have an essential role in this vision and it is our job to make it possible.

In this project, we have developed a reliable framework that allows reasoning across distributed belief networks. Based on Bayesian networks, this framework maintains coherence and consistency between the beliefs of different nodes, as well as it is able to handle uncertainty inherent to hardly any type of knowledge. The developed tool is scalable, stable, tolerant to communication failures, and portable. It uses asynchronous communication to maintain the independence between nodes. Thus, each node in the belief network is autonomous, and can reason with all the information it get, both its own information and external data obtained from other nodes.

Nevertheless, other improvements can be done to this framework. In this project, we have presented several functionalities that have not been totally included in the resulting framework. These pending features have been introduced in this report, and form the basis for future work in this field. Some of this future work is summarized below.

The developed framework performs the detection of cycles in the computed global graph. The continuation to this work should make sure a correction of situations where cycles appear is done, allowing the operation of this framework in more complex and dynamic systems.

Before the propagation of beliefs between different nodes takes place, their initialization is needed. Currently, this initialization is performed following a hierarchical structure. To give more realistic and accurate results, it would be necessary to implement a belief initialization technique to weigh the knowledge of each node in terms of the relevance of the node.

In conclusion, this distributed reasoning framework is able to adapt to different scenarios or situations. This framework can be used in a huge variety of applications in which distributed reasoning can provide a great advantage. An application of this framework to a concrete scenario is presented in this report. The application in FTTH scenarios serves as an example for future developments. It remains as future work, the use of this framework in other scenarios to bring out the potential application field this framework can have.

Bibliography

[1] J. G. O. Álvaro Carrera, J. Garcia-Algarra, P. Arozarena, and M. Garijo, "A Multi-Agent System with Distributed Bayesian Reasoning for Network Fault Diagnosis," in *Advances on Practical Applications of Agents and Multiagent Systems: 9th International Conference on Practical Applications of Agents and Multiagent Systems*, 2011, vol. 88, p. 113.

[2] D. Li and Y. Du, *Artificial intelligence with uncertainty*. Tsinghua University; Beijing. China: CRC Press, 2008.

[3] D. B. Leake, "Case-based reasoning," *The Knowledge Engeneering Review*, vol. 9, pp. 196-197, Jan. 1994.

[4] D. Zhang, "Multi-agent based control of large-scale complex systems employing distributed dynamic inference engine," 2010.

[5] F. Hayes-Roth, "Rule-based systems," *Communications of the ACM*, vol. 28, no. 9, pp. 921–932, 1985.

[6] L. Zadeh, "Fuzzy logic," *Computer*, 1988.

[7] U. B. Kjærulff and A. L. Madsen, *Bayesian Networks and Influence Diagrams*. Springer New York, 2008, pp. 3-15.

[8] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[9] G. Pavlin, M. Mans, and J. Nunnink, "An agent-based approach to distributed data and information fusion," in *Intelligent Agent Technology, 2004.(IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, 2004, pp. 466–470.

[10] M. A. Paskin and C. E. Guestrin, "Robust probabilistic inference in distributed systems," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 2004, pp. 436–445.

[11] K. B. Laskey, P. Costa, and T. Janssen, "Probabilistic ontologies for knowledge fusion," in *Information Fusion, 2008 11th International Conference on*, 2008, pp. 1–8.

[12] F. Jensen, "Bayesian networks and influence diagrams," *Risk Management Strategies in Agriculture; Huirne et ….*

[13] F. V. Jensen and T. D. Nielsen, *Bayesian networks and decision graphs*. Springer Verlag, 2007.

[14] S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen, "HUGIN—a shell for building Bayesian belief universes for expert systems," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, vol. 2, pp. 1080–1085.

[15] L. Getoor and B. Taskar, "Introduction to statistical relational learning," *MIT Press*, 2007.

[16] F. Jensen, “An introduction to Bayesian networks,” 1996.

[17] P. P. Shenoy, “Binary join trees for computing marginals in the Shenoy-Shafer architecture,” *International Journal of Approximate Reasoning*, vol. 17, no. 2-3, pp. 239–263, 1997.

[18] A. Darwiche, “Recursive conditioning,” *Artificial Intelligence*, 2001.

[19] P. Costa and M. Ladeira, “A first-order bayesian tool for probabilistic ontologies,” *Proceedings of the 21st …*, 2008.

[20] Y. Xiang and V. Lesser, “Justifying multiply sectioned Bayesian networks,” in *icccn*, 2000, p. 0349.

[21] Z. Ras, M. Zemankova, and Y. Xiang, *Distributed multi-agent probabilistic reasoning with Bayesian networks - Methodologies for Intelligent Systems - Lecture Notes in Computer Science*, vol. 869. Springer Berlin / Heidelberg, 1994, pp. 285-294-294.

[22] Y. Xiang, “Distributed scheduling of multiagent communication,” in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995, p. 390.

[23] Y. Xiang, “Verification of DAG structures in cooperative belief network-based multiagent systems,” *Networks*, vol. 31, no. 3, pp. 183–191, 1998.

[24] X. An and N. Cercone, “Iterative multiagent probabilistic inference,” in *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, 2006, pp. 240–246.

[25] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, “MASON: A multiagent simulation environment,” *Simulation*, vol. 81, no. 7, p. 517, 2005.

[26] S. Company, “Maven: the definitive guide,” 2008.

Glossary

TERM	CONCEPT
BN	Bayesian Network
CPT	Conditional Probability Table
DAG	Directed Acyclic Graph
DPN	Distributed Perception Network
FOL	First-Order Logic First-order logic is distinguished from propositional logic by its use of quantifiers. Each interpretation of first-order logic includes a domain of discourse over which the quantifiers range.
FTTH	Fiber to the Home Scenario in which fiber is available directly to the customer's home provided by GPON directly from the CO (Central Office)-based OLT.
GPON	Gigabit-capable Passive Optical Network Standard that support high rates, enhanced security, and choice of Layer 2 protocol (ATM, GEM, Ethernet).
HAN	Home Area Network
IA	Intelligent Agent An autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals.
JPD	Joint Probability Distribution
JT	Junction Tree
LAN	Local Area Network
MAS	Multi-Agent System A system composed of multiple interacting intelligent agents.
MEBN	Multiply Entity Bayesian Network A first-order probabilistic logic that combines the representational power of first-order logic and Bayesian networks.
MFrags	MEBN fragments
MSBN	Multiply Sectioned Bayesian Network
MSDAG	Multiply Sectioned DAG
MTheories	MEBN Theories A set of MFrags collectively satisfies consistency constraints ensuring the existence of a unique joint probability distribution over instances of the random variables mentioned in the MFrags.

OLT	Optical Line Terminal Active device which serves as the service provider endpoint of a passive optical network. It performs conversion between the electrical signals used by the service provider's equipment and the fiber optic signals used by the passive optical network and coordinate the multiplexing between the conversion devices on the other end of that network (ONTs).
ONT	Optical Network Terminal Active device used to terminate the fiber optic line, demultiplex the signal into its component parts (voice telephone, television, and Internet), and provide power to customer telephones.
OWL	Web Ontology Language Family of knowledge representation languages for authoring ontologies. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms.
PLDM	Prior / Likelihood Decomposable Models
PR-OWL	Probabilistic OWL
RPC	Remote Procedure Call
SAS	Single-Agent System A system composed of a single intelligent agent.
WDM	Wavelength-Division Multiplexing A technology multiplexes a number of optical carrier signals onto a single optical fiber by using different wavelengths of laser light. This technique enables bidirectional communications over one strand of fiber, as well as multiplication of capacity.

Appendix 1. Developer manual

This chapter is intended to present the project thinking on a developer that continues the work done so far. Thus, this manual is intended as an initial reference point and reference for future iterations that new developers join the project with the consequent lack of diagnostic system present in multi-agent project.

1. Subversion

All developed source has been saved and organized using the subversion plugin integrated with eclipse (subclipse).

Each change done has been documented with the corresponding information written in English language.

Inside a commentary, the first words are always the more relevant ones while last words are other interesting details about the performed changes. The convention used is that the first sentences always indicates the state of the code. It shows if the code is completely working or if it has any detected bug or unsolved situation that needs to be specially cared by the developer.

Subversion repository used has been Fibit project repository. Inside *trunk* folder, *MSBN* folder contains all developed source about this project.

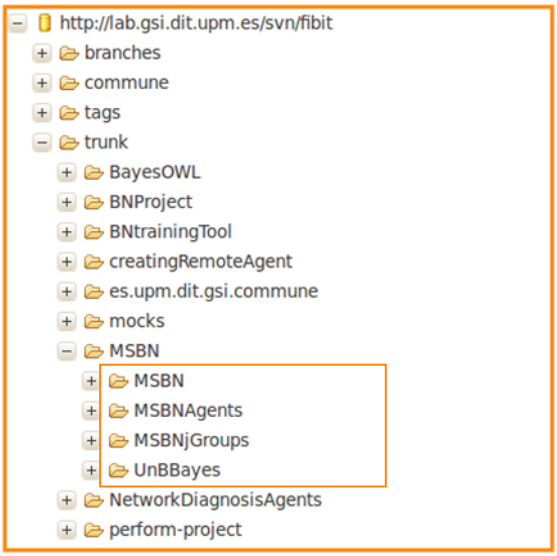


Figure VI-5 Subversion directory structure

As UnBBayes has not been included in Maven repositories yet, the whole UnBBayes main project has been uploaded to Fibit subversion repository. Only a little change has been done over this project, which is the modification of *pom.xml* maven file to allow the skipping of UnBBayes tests during the installation of this in local Maven repository.

Inside MSBN folder, the version of this project that uses synchronous architecture is contained. The important classes are in the path *src/main/java/unbbayes/prs/msbn*. Test classes and examples are in the path *src/main/java/unbbayes/example/multi*, while *single* folder contains examples for the original MSBN version (Single-agent) provided by UnBBayes.

MSBNAgents folder contains the agent framework developed for multi-agent synchronous architecture using Jadex agent platform.

Finally, MSBNjGroups folder contains the multi-agent iterative MSBN framework developed as result of this project. Despite it also contains *single* and *multi* versions, *multiIterative* folders are those that contain the multi-agent iterative MSBN framework. The most important classes can be found in the path *src/main/java/unbbayes/prs/msbn/multi_iterative*. In addition, examples and tests can be found in the path *src/main/java/unbbayes/example/multi_iterative*.

2. Maven

All projects developed use Maven to automate compilation and building of the project, as well as dependencies management [26].

To use this tool, we have used the Maven plugin for eclipse *m2eclipse*, which let as saving some time and efforts.

As UnBBayes is not part of Maven repositories yet, the first project that needs to be installed is UnBBayes project. For that purpose, we must download or import UnBBayes' project using subclipse and click on *run as / maven install* in the secondary menu of the project.

After UnBBayes' project is installed in the local Maven repository, all other project can be installed by proceeding the same way.

Appendix 2. Installation manual

1. Install Java JDK6

Windows:

1. Download and install Java JDK 6: JDK6. A 32bit version is needed for JpCap to work.
2. Set 'JAVA_HOME' environment variable to point to installation directory. *For example: 'C:\Program Files\Java\jdk1.6.0_25'.*
3. Add '%JAVA_HOME%\bin' to 'path' environment variable.

Ubuntu:

1. Uninstall Open JDK if it's installed in your system.
2. Download and install Java JDK 6: JDK6. A 32bit version is needed for JpCap to work.
3. Add this location to the path by modifying '.bashrc' file located in your home folder. For example, the following lines could be added to '.bashrc' file:

```
export JAVA_HOME=/home/username/Descargas/JAVA/jdk1.6.0_23
export PATH=$JAVA_HOME/bin:$PATH
```

2. Install Maven

Windows:

1. Download the last version of Maven2 .zip from Maven_2.2.1.
2. Unzip it in the location where you want to have it installed.
3. Modify the path by adding the location of the bin directory located inside the unzipped folder. *For example: C:\Program Files\apache-maven-2.2.1\bin.*
4. Test Maven2 runs correctly by writing 'mvn' in the console.

Ubuntu:

Go to Synaptics Package Manager and install *maven2*.

3. Install Eclipse and proper plugins

1. Download and install Eclipse Classic: Eclipse
2. Set in .ini file in eclipse folder, the launching option

```
-vm
C:/...path_to_java_jdk.../bin
```

3. Install **Subclipse_1.4** plugin: Subclipse_1.4
4. Install **m2eclipse** plugin: m2eclipse

4. Prepare the project and install dependencies:

1. Download UnBBayes project from svn:
   `http://lab.gsi.dit.upm.es/svn/fibit/trunk/MSBN/UnBBayes/UnBBayes`
2. Download MSBN project from svn:
   `http://lab.gsi.dit.upm.es/svn/fibit/trunk/MSBN/MSBNjGroups`
3. Execute `Maven Install` over the whole project of UnBBayes.
4. Execute `Maven Install` over the whole project of MSBN.