

A Project Report on
**IMPLEMENTATION OF BAYESIAN NETWORK USING
JUNCTION TREE ALGORITHM**

Submitted in partial fulfillment of the requirement for the Eighth Semester

Bachelor of Engineering
In
Computer Science and Engineering
Of
Visvesvaraya Technological University, Belgaum



Submitted by

AKARSH M.P	[1DS11CS005]
KENNETH RITHVIK	[1DS11CS044]
NAGESH	[1DS11CS056]
PAVAN KUMAR A	[1DS11CS067]

Under the guidance of
Mrs. Preeti Satish
Associate Professor,
Dept. of CSE, DSCE



2014-2015

Department of Computer Science and Engineering,
DAYANANDA SAGAR COLLEGE OF ENGINEERING
BANGALORE – 560078

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore-560078.

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the project report entitled “**Implementation of Bayesian Network using Junction Tree Algorithm**” is a bonafide work carried out by **AKARSH M.P [1DS11CS005]**, **KENNETH RITHVIK [1DS11CS044]**, **NAGESH [1DS11CS056]** and **PAVAN KUMAR A [1DS11CS067]** in partial fulfillment for the VIII SEM, Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2014-15. It is certified that all corrections or suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project prescribed for the Bachelor of Engineering Degree.

Mrs.Preeti Satish
(Internal Guide)
Associate Professor
Dept. of CSE, DSCE

Dr. Ramesh Babu D.R
Vice Principal & HOD,
Dept. of CSE, DSCE.
Bangalore.

Dr. K. Karibasappa
Principal,DSCE,
Bangalore.

Name of the Examiners

Signature with date

1._____

2._____

ACKNOWLEDGEMENT

Any accomplishment requires the effort of many people and this work is no different. We are indebted to **LDRE (Electronics Radar Development Establishment), Bangalore**, for offering us the opportunity to train in their campus between Jan-Apr 2015. Training in LRDE was a constant learning process for all of us. We learnt something new every day. This was possible only because of the ready help we got from the members of the SQAG (Software Quality Assurance Group) team.

Enough thanks cannot be mentioned to **Mr. Justin Sagayaraj, Scientist F, SQAG, DRDO** who in spite of his hectic schedule found time to help us with our project.

We would like to express our most sincere gratitude to our guide, **Mrs. Preeti Satish**, Associate Professor, Department of Computer Science and Engineering, for her excellent guidance which proved to be imperative for the completion of this project.

We express our sincere gratitude to **Dr. Ramesh Babu. D. R**, HOD, Department of Computer Science & Engineering, DSCE for his constant support and encouragement.

We also express our sincere gratitude to **Dr. Karibasappa**, Principal, DSCE, for his advice and guidance that helped us to complete this project with success.

We thank our institution, **Dayananda Sagar College of Engineering** without which our project would have never seen the light of completion.

Akarsh M.P	[1DS11CS005]
Kenneth Rithvik	[1DS11CS044]
Nagesh	[1DS11CS056]
Pavan Kumar A	[1DS11CS067]

ABSTRACT

A **Bayesian network**, **Bayes network**, **belief network**, **Bayes(ian) model** or **probabilistic directed acyclic graphical model** is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are DAGs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes that are not connected represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables, and gives (as output) the probability (or probability distribution, if applicable) of the variable represented by the node.

The **junction tree algorithm** (also known as 'Clique Tree') is a method used in machine learning to extract marginalization in general graphs. In essence, it entails performing belief propagation on a modified graph called a junction tree. The basic premise is to eliminate cycles by clustering them into single nodes.

CONTENTS

1. INTRODUCTION	1
1.1. What are Bayesian Networks?	1
1.2. Organizational Profile	2
1.2.1. A Brief Historical Background	2
1.2.2. Related to our project	3
1.3. Problem statement	5
1.3.1. Existing System	5
1.3.2. Proposed System	6
2. LITERATURE SURVEY	7
3. PROPOSED SYSTEM.....	9
3.1. Converting BN to probability.....	10
4. SOFTWARE REQUIREMENTS AND SPECIFICATION	11
4.1. System Requirements	11
4.1.1. Hardware Requirements	11
4.1.2. Software Requirements	11
4.2. Requirement Analysis	12
4.2.1. Functional Requirements	12
4.2.2. Non-Functional Requirements	12
5. SYSTEM DESIGN	13
5.1. High Level Design Diagrams	14
5.2. Sequence Diagram.....	16
5.3. Use Case Diagram.....	17
5.4. Data Flow Diagram	17
5.4.1. Level 0 and Level 1 Data Flow Diagrams	18

6. IMPLEMENTATION	19
6.1. Java Technology.....	19
6.2. The Java Program language.....	19
6.3. Java Technology.....	20
6.4. What can Java Technology Do?.....	21
6.5. How will Java Technology change my life?.....	23
6.6. Class Diagram.....	24
 7. TESTING	 25
7.1. Levels of Testing	25
7.1.1. Unit Testing	25
7.1.2. Integration Testing	25
7.1.3. System Testing	26
7.2. Types of Testing	26
7.2.1. Acceptance Testing	26
7.2.2. correctness Testing	26
7.2.4. Black Box Testing	27
7.2.3. White Box Testing	28
7.2.3. Performance Testing	29
7.2.5. Reliability Testing	30
7.2.6. Security Testing	30
7.3. Unit Testing for Main Modules	31
7.3.1. Unit Testing of creating network.....	31
7.3.2. Unit Testing of entering evidence	32
7.3.3. Unit Testing for Quering nodes	32
 8. RESULTS	 33
8.1. Network Editor	45
8.2. Querying the network.....	46

9. SOURCE CODE.....	39
10.CONCLUSION	59
11. REFERENCES	60

LIST OF FIGURES AND TABLES

LIST OF FIGURES

Sl. No	Fig. No	Figure Name	Pg. No
1	Fig 1.1	Mobile Radar System	03
2	Fig 1.2	Stationary Radar System	03
3	Fig 1.3	IFF Radar System	04
4	Fig 3.1	Structure of Bayesian Network	09
5	Fig 5.1	High Level Design	14
6	Fig 5.2	Sequence Diagram	16
7	Fig 5.3	Use case Diagram	17
8	Fig 6.1	Java Virtual Machine	20
9	Fig 6.2	Java Compiler	20
10	Fig 6.3	Running Java Platform	21
11	Fig 6.4	Java2 SDK	23
12	Fig 6.5	Class Diagram	24
13	Fig 8.1	Console frame	33
14	Fig 8.2	Editor frame	33
15	Fig 8.3	Creating a network	34
16	Fig 8.4	Opening a network file	34
17	Fig 8.5	Enemy aircraft identifier network	35
18	Fig 8.6	Entering probable distribution	35
19	Fig 8.7	Editing network properties	36
20	Fig 8.8	Providing evidence	36
21	Fig 8.9	Querying a node	37
22	Fig 8.10	Bayesian network query	37
23	Fig 8.11	Saved network in XML format	38

LIST OF TABLES

Sl. No	Table. No	Table Name	Pg. No
1	Table 3.1	CPT	10
2	Table 7.1	Creating a network	31
3	Table 7.2	Entering probabilistic value	31
4	Table 7.3	Setting evidence value	32
5	Table 7.4	Querying a node	32

CHAPTER 1

INTRODUCTION

1.1. What are Bayesian networks?

Bayesian networks (BNs), also known as belief networks (or Bayes nets for short), belong to the family of probabilistic graphical models (GMs). These graphical structures are used to represent knowledge about an uncertain domain. In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Hence, BNs combine principles from graph theory, probability theory, computer science, and statistics.

Graphical Models with undirected edges are generally called Markov random fields or Markov networks. These networks provide a simple definition of independence between any two distinct nodes based on the concept of a Markov blanket. Markov networks are popular in fields such as statistical physics and computer vision.

Because a Bayesian network is a complete model for the variables and their relationships, it can be used to answer probabilistic queries about them. For example, the network can be used to find out updated knowledge of the state of a subset of variables when other variables (the evidence variables) are observed.

A Bayesian network can thus be considered a mechanism for automatically applying Bayes' theorem to complex problems.

Because a Bayesian network is a complete model for the variables and their relationships, it can be used to answer probabilistic queries about them. For example, the network can be used to find out updated knowledge of the state of a subset of variables when other variables (the evidence variables) are observed. This process of computing the posterior distribution of variables given evidence is called probabilistic inference. The posterior gives a universal sufficient statistic for detection applications, when one wants to choose values for the variable subset which minimize some expected loss function, for instance the probability of decision error. A Bayesian network can thus be considered a mechanism for automatically applying Bayes' theorem to complex problems

1.2. Organizational profile

A brief introduction of LRDE (Electronics and Radar Development Establishment) is given along with their achievements in the past and their areas of work. Recognition in LRDE is also mentioned and also the systems and Radar related to our project.

1.2.1. A brief historical background

Electronics and Radar Development Establishment (LRDE) is one of the R&D Establishments set up under the Defense R&D Organization to address the services needed in the field of radar, Communication systems and related technologies.

The genealogy of LRDE originates from the Inspectorate of Scientific Stores created in 1939 at Rawalpindi. This was re-christened as TDE (Instruments & Electronics) in early 1946 and relocated in Dehradun. With electronic industrial complexes sprouting in the salubrious Bangalore outskirts, the Electronics part of TDE (I&E) branched off to settle in this seminal city in 1955. Coinciding with the promotion of DRDO, the then TDE (E) was bifurcated into Inspectorate of Electronics Equipments (ILE) and LRDE on 01 Jan 1958 to meet the aspiration of Defense Services. LRDE moved to the present serenely location in 1986 to find place for expansion and growth.

LRDE has made very significant contributions towards indigenous design and development of complex military electronic systems communication and radar which are serving our armed forces. Today LRDE has the core competence and expertise to build world class radar systems and software driven electronic systems.

Vision of LRDE: To create a centre of excellence in design and development of radars and related Technologies.

Mission of LRDE: Develop Radar systems and related technologies to cater to the needs of Services. Enhance the infrastructure, knowledge base and technologies for achieving self reliance.

1.2.2. Related to our project



Fig 1.1: Mobile Radar System

This Radar System is designed for detection of intruding aircraft including low level attacks. This Radar helps in the detection of whether the aircraft is enemy or friendly. These kinds of Radars are generally known as IFF (Identify Friend or Foe) Radars. These Radars work on the principle that initially the radar is made to send a signal to the unknown aircraft and based on the reply of the aircraft it detects whether the aircraft is a friend or foe. Some of the main features are automated Track While Scan (TWS), integrated IFF and high scan rate for high speed target detection.

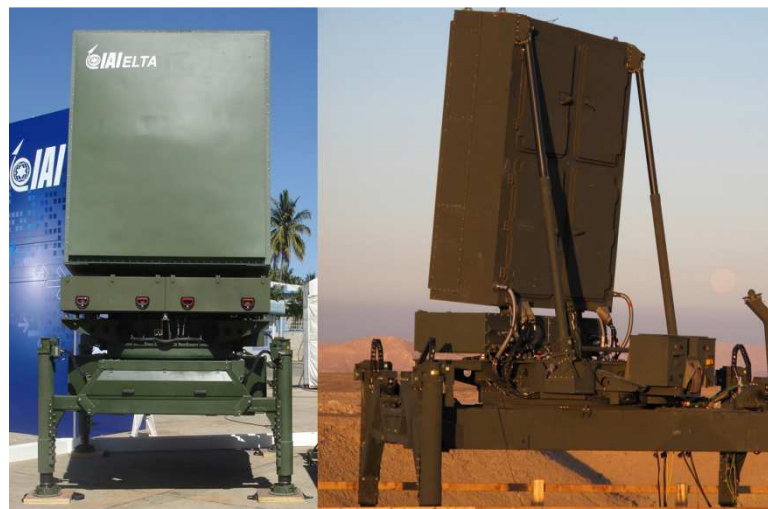


Fig 1.2: Stationary Radar System

In contrast to the older AN/MPQ-53 Passive Electronically Scanned Array (PESA) radar set of the MIM-104 Patriot PAC-2, the Green Pine is a Active Electronically Scanned array (AESA) solid state radar, Stages of missile interception by the Arrow system, using Green Pine radar.

In contrast to the older AN/MPQ-53 Passive Electronically Scanned Array (PESA) radar set of the MIM-104 Patriot PAC-2, the Green Pine is a active electronically scanned array (AESA) solid state radar. Unlike the advanced AN/TPY-2 X band radar of the Terminal High Altitude Area Defense system, Green Pine operates at L band - in the range 500 MHz to 1,000 MHz ,or 1,000 MHz to 2,000 MHz

Green Pine reportedly operates in search, detection, tracking, and missile guidance modes simultaneously, capable of detecting targets at ranges of up to about 500 km (310 mi), and is able to track more than 30 targets at speeds over 3,000 m/s (10,000 ft/s). It discriminates targets from natural clutter and countermeasures, illuminates the true target and guides the missile to within 4 m (13 ft) of the target.

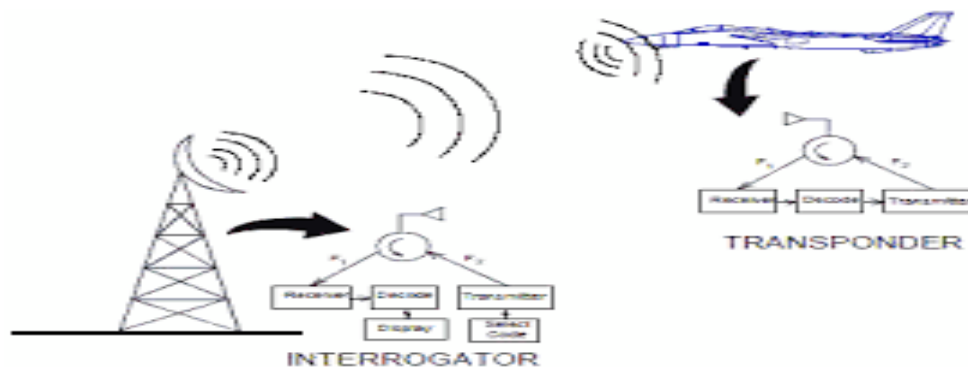


Fig 1.3: IFF Radar System

The above figure shows a simple communication between the radar and an aircraft. In this figure the radar also known as the interrogator sends or transmits a signal to the aircraft. If the aircraft is a friendly aircraft the transponder in the aircraft will receive the signal and reply back via the transponder back to the interrogator saying that it is friendly aircraft. If the aircraft is an enemy aircraft they (generally) will not respond and thus it will be treated as an enemy aircraft thus able to successfully detect whether the identified unknown aircraft is a friend or foe.

1.3. Problem Statement

Bayesian networks (BNs), belong to the family of probabilistic graphical models (GMs). These graphical structures are used to represent knowledge about an uncertain domain. In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Hence, BNs combine principles from graph theory, probability theory, computer science, and statistics.

There exists a lot of scope for this model to be used effectively in the field of defense where uncertainties may be involved. This model is especially relevant here as there involves a lot of uncertainty while making decisions in the battle ground. Helping a computer system to automatically or artificially take decisions during these circumstances will augment the capabilities human element in battle. Prior evidence obtained or known can also be used to improve the accuracy of the decision taken.

1.3.1. Existing Systems

The following are the currently existing systems, related to Bayesian networks.

- MSBNx Editor and Toolkit-

MSBNx is a component-based Windows application for creating, assessing, and evaluating Bayesian Networks, created at Microsoft Research. The application's installation module includes complete help files and sample networks. Bayesian Networks are encoded in an XML file format. The application and its components run on Windows 98, Windows 2000, and Windows XP.

- Netica Application-

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").

- Pattern Analysis Tools-

HUGIN Expert A/S is a leading provider of model-based decision support software for reasoning under uncertainty. HUGIN software tools are based on Bayesian Network and influence diagram technology, an advanced statistical technique that supports decision-making in complex domains on the basis of partial, uncertain or unknown information.

Disadvantages:-

- Commercial software that cannot be used without a license
- They are large and complex packages that need a lot of resources to run.
- They are not custom made for military applications.

1.3.2. Proposed System

Because a Bayesian network is a complete model for the variables and their relationships, it can be used to answer probabilistic queries about them. For example, the network can be used to find out updated knowledge of the state of a subset of variables when other variables (the evidence variables) are observed. This process of computing the posterior distribution of variables given evidence is called probabilistic inference. The posterior gives a universal sufficient statistic for detection applications, when one wants to choose values for the variable subset which minimize some expected loss function, for instance the probability of decision error. A Bayesian network can thus be considered a mechanism for automatically applying Bayes' theorem to complex problems. This software has been closely developed with a defense establishment and hence is very much suited for battle deployment.

Advantages:-

- Developed only for the defense sector and hence does not need any license for it to be used by them.
- A small and simple software package that does not require a lot of resources to run.
- Can be applied in real-time defense equipment.
- Highly platform independent and portable software.

CHAPTER 2

LITERATURE SURVEY

- Title: Hierarchical junction trees as the secondary structure for inference in Bayesian Networks.

Abstract: Traditionally, a single junction tree is used as the secondary structure for inference in a Bayesian network. However, its applicability and efficiency are restricted by the size of the junction tree. In this paper, we demonstrate that using a hierarchy of junction trees (HJT) as the secondary structure instead will greatly alleviate this restriction and improve the performance. We also compare the proposed HJT with other similar schemes for inference in Bayesian networks.

- Title: Bayesian Estimation vs. Fuzzy Logic for heuristic reasoning.

Abstract: Bayesian estimation theory and fuzzy logic are used to derive knowledge combination rules for heuristic search algorithms. Such algorithms are typically used with expert systems. Heuristics are used to select candidate solutions or partial solutions of a complex problem whose solution space is too large to be fully explored. The information coming from the various heuristics and from observations made during the search must be combined. Combination and decision rules are first derived based on a probabilistic approach. Then, a fuzzy logic approach is followed and compared with the first approach.

- Title: Preliminary study of the progress on using Bayesian Networks for educational assessment.

Abstract: Educational assessment has been subject to numerous studies in recent years. The emergence of Bayesian Networks (BN) as a powerful probabilistic inference tool has been integrated by many into researches on modeling the education domain to measure technical as well as cognitive performance of a student. We have seen many different subjective perspectives in interpreting the educational context, such as measuring individual educational backgrounds, emotional state, teaching methods, financial background, students' learning styles, specific academic concept

skills and many other abstract variables. Comparatively, there hasn't been a widely-accepted Bayesian Network model in the domain of educational assessment and we are in need of one. This paper provides a preliminary study of the progress of research on educational assessment methods using Bayesian Networks. In this summarized survey of literatures we review and compile the recent trends of how researchers have applied and implemented Bayesian Networks to solve the complex problem of educational assessment in their own environments.

- Title: Dynamic Discretization: A Combination Approach.

Abstract: Supervised discretization refers to the problem of transforming continuous attributes of a decision table into discretized ones. It is important for some artificial intelligence theories where nominal data are required or preferred. Instead of depending on the experience of human experts, supervised discretization algorithms learn from the data. However, the results of such algorithms may be sensitive to the change of the data. In this paper, we propose to compute more stable and informative discretization schemes through suitable sampling and scheme combination. Discretization schemes computed in this way are called dynamic discretization schemes. Experimental results on some well-known datasets show that they are helpful for obtaining decision rules with better accuracy and F-measure.

- Title: Structure learning of BN by genetic algorithms.

Abstract: We present a new approach to structure learning in the field of Bayesian networks. We tackle the problem of the search for the best Bayesian network structure, given a database of cases, using the genetic algorithm philosophy for searching among alternative structures. We start by assuming an ordering between the nodes of the network structures. This assumption is necessary to guarantee that the networks that are created by the genetic algorithms are legal Bayesian network structures. Next, we release the ordering assumption by using a “repair operator” which converts illegal structures into legal ones. We present empirical results and analyze them statistically. The best results are obtained with an elitist genetic algorithm that contains a local optimizer.

CHAPTER 3

PROPOSED METHOD

3.1. Converting BN to Probability Table

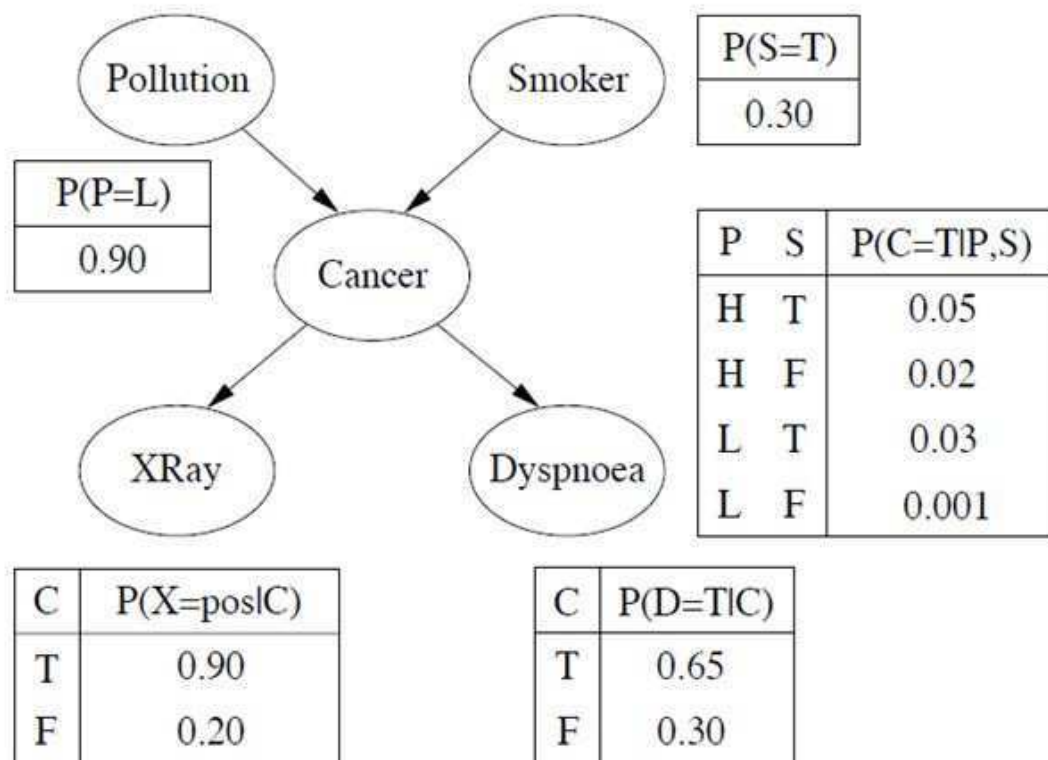


Fig 3.1: Structure of Bayesian Network

The first step is to get the input, the probability of each event occurring with true or false values. After that the Bayesian network is created which is a directed acyclic graph and the relation between each of the nodes is depicted as shown in figure 3.1. This figure comprises of a simple 5 node interaction where the probability of cancer is being found.

As the above figure shows for example the probability for the X-ray to show the Cancer actually exists will be .90 correctly. This means that there is a .20 chance that it will show a negative or false when it actually might exist. This means that there are chances that it might also show a false positive or false negative. So for every node considering all these possibilities we construct a conditional probability table.

Our goal is to construct a probability table for this Bayesian network. This can be achieved by using the Junction Tree Algorithm. Our proposed method will give us the probability for a particular node when we click on it.

The conditional probability table will look as follows:

Node P(S)=0.3	No Evidence	Reasoning Case				
		Diagnostic D=T	Predictive S=T	Intercausal		Combined D=T S=T
				C=T	C=T S=T	
Bel(P=high)	0.100	0.102	0.100	0.249	0.156	0.102
Bel(S=T)	0.300	0.307	1	0.825	1	1
Bel(C=T)	0.011	0.025	0.032	1	1	0.067
Bel(X=pos)	0.208	0.217	0.222	0.900	0.900	0.247
Bel(D=T)	0.304	1	0.311	0.650	0.650	1

Table 3.1: Conditional Probability Table

As the table shows the first row indicates that the belief of pollution is high. For this we calculate the remaining probabilities. The third column shows or is constructed on the basis that dyspnoea is true. Similarly the remaining columns consider some of the nodes to be true and calculate the probability of the remaining nodes.

This shows that we can start calculating the probability of whichever node we want either from the causes or from the way we are going to detect it. This is only for this example since this example has causes and ways to detect. The evidence column shows the probability calculation for that particular node.

Hence in our proposed method when we click on the particular node of which we have to calculate the probability, the probability values for that node will be shown after considering the probabilities of all its related nodes.

Since it's an acyclic directed graph the initial nodes whose in degree is 0 should be fed with a data set values to calculate the probability of their following nodes.

CHAPTER 4

SYSTEM REQUIREMENT SPECIFICATION

A Software Requirements Specification (SRS) is a complete description of the behavior of the system to be developed. It includes the functional and non functional requirement for the software to be developed. The functional requirement includes what the software should do and non functional requirement include the constraint on the design or implementation of the system. Requirements must be measurable, testable, related to identified needs or opportunities, and defined to a level of detail sufficient for system design.

The writing of software requirement specification reduces development effort, as careful review of the document can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct. The SRS discusses the product but not the project that developed it; hence the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

4.1. System Requirements

4.1.1. Hardware Requirements

- Intel Pentium IV processor.
- 4 GB of RAM required.
- Disk space of 1 GB required.

4.1.2. Software Requirements

- Operating system required is Windows XP/7.
- Development tool used is Netbeans.
- The Development environment used is JDK version 8
- The back end language used is JAVA, J2EE.

4.2. Requirement Analysis

Requirements Analysis is the process of understanding the customer needs and expectations from a proposed system or application and is a well-defined stage in the Software Development Life Cycle model. Requirements are a description of how a system should behave or a description of system properties or attributes. It can alternatively be a statement of 'what' an application is expected to do. The Software Requirements Analysis Process covers the complex task of eliciting and documenting the requirements of all these users, modeling and analyzing these requirements and documenting them as a basis for system design.

4.2.1. Functional Requirements

- Design and Development of a user friendly interface so that the user can create a network.
- Design and Development of a parser and loader that can load saved networks.
- Design and Development of an interface to show the results of the inference on the network.
- Design and Development of a help console to aid and guide the user while using the software.
- Design and Development of an interface to edit the nodes and the network.
- Design and Implementation of the junction tree algorithm to perform inference on the Bayesian network.
- Design and Development of a common format to save the existing network so that it can be transferred or used again.

4.2.2. Non-Functional Requirements

- Reliability, Availability and Maintainability are three important requirements in Real Time Scenarios.
- Performance – The performance of the proposed approach must be better than the existing ones.
- Flexibility of the program(should be able to update it later/add more categories or users)
- Efficiency – The efficiency of the approach is important since the number of nodes we are using also determines the computation rate.

CHAPTER 5

SYSTEM DESIGN

The purpose of the design is to plan the solution of a problem specified by the requirements document. This phase is the first step in moving from problem to the solution domain.

The design of the system is perhaps the most critical factor affecting the quality of the software and has a major impact on the later phases, particularly testing and maintenance. System design aims to identify the modules that should be in the system, the specification of these modules and to interact with each other to produce the required results. At the end of system design all the major data structures, file formats, output formats, as well as major modules in the system and their specifications are decided.

At this stage in the design process, the designer should know the organization of the design and what each function should do. Design entity description is concerned with producing a short design specification for each design. This describes the function, its inputs and its outputs.

The best way to manage functional descriptions is to maintain them in the data dictionary. Most components described in the system architecture section will require a more detailed discussion.

Other low level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed of a system software component. An efficient design would be complete with:

- Architectural diagram: This gives brief information about the modules used and structure of the model. This is called high level design (level 0).
- Control flow diagram (CFD): Different functions used in the program and how the control flows between these functions.
- Data flow diagram (DFD): Data flows in between the module. Both CFD and DFD together constitutes low level design (level 1).
- Activity diagram: It shows the sequence of steps that make up a complex process.

5.1. High Level Design Diagram

Top level design provides the overview design which does not include the detailed design. Top level design is also called as level 0 design diagram, which gives overall process and modules involvement in the project. This is context level top level design is to produce a level 0 diagram displays some of the detail diagram, which gives overall process and modules involvement in the project. An internal data store that must be present in order to do its job is identified, and flow of data between the various parts of system is being showed.

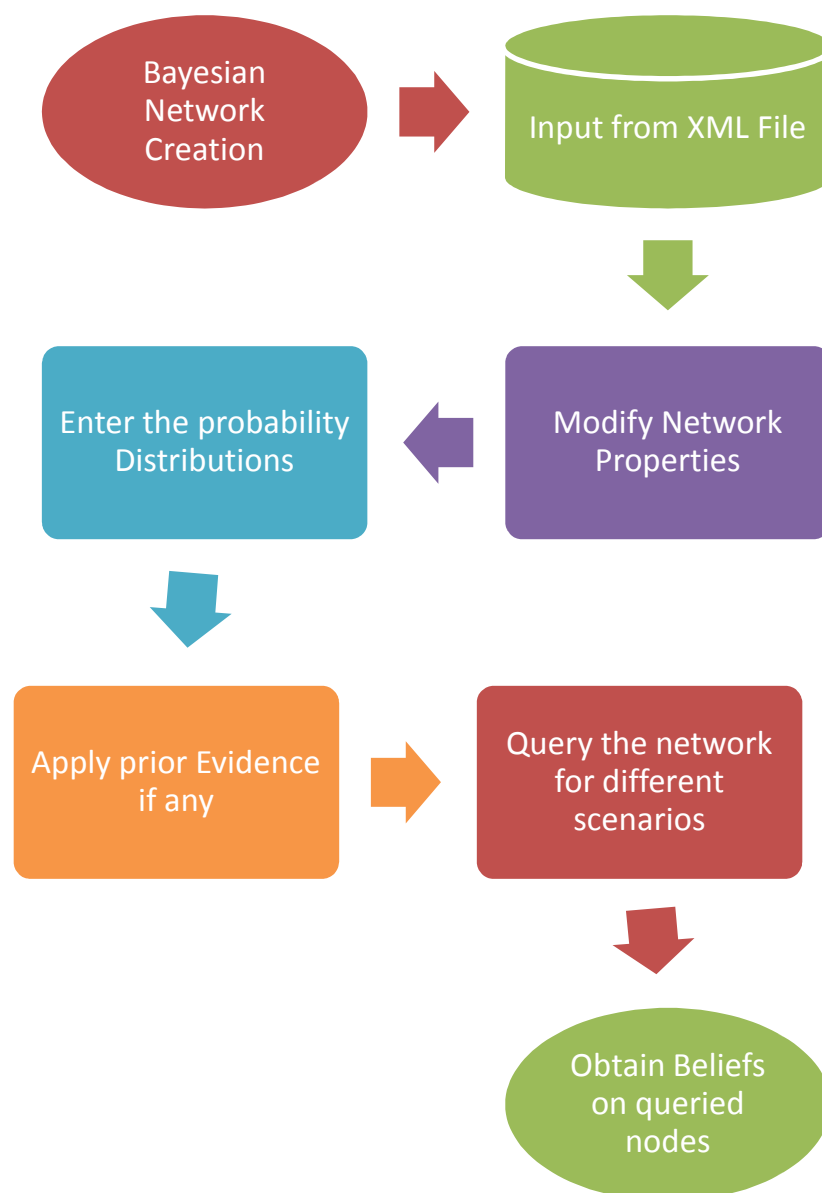


Fig 5.1 High level Design

Create Bayesian Network

In this stage we can use the interface to create the static Bayesian Network. The structure of the network is studied in this stage and the dependencies that each node might have on another node is also figured out.

Input from XML File

As an alternative we can also load a predefined Bayesian Network that might have been constructed previously and saved. This enables us to save and use complex networks later.

Modify Network Properties

Any changes to the structure of the network can be done at this stage.

Enter the Probability Distributions

This stage is an important stage where we can actually define how each probability variable or node is dependent on other using quantitative values. This includes the conditional probability table. This is called the probability distribution.

Apply prior Evidence if any

At this stage we can input any prior evidence we might have about the value a particular probability variable takes. This helps greatly in simplifying the network and making the results more realistic according to the current scenario.

Query the network for different Scenarios

At this stage we can query or instruct the software to give us the beliefs of particular variables.

Obtain the Results

At this stage we obtain the beliefs or the probabilities of a particular event occurring.

5.2. Sequence Diagram

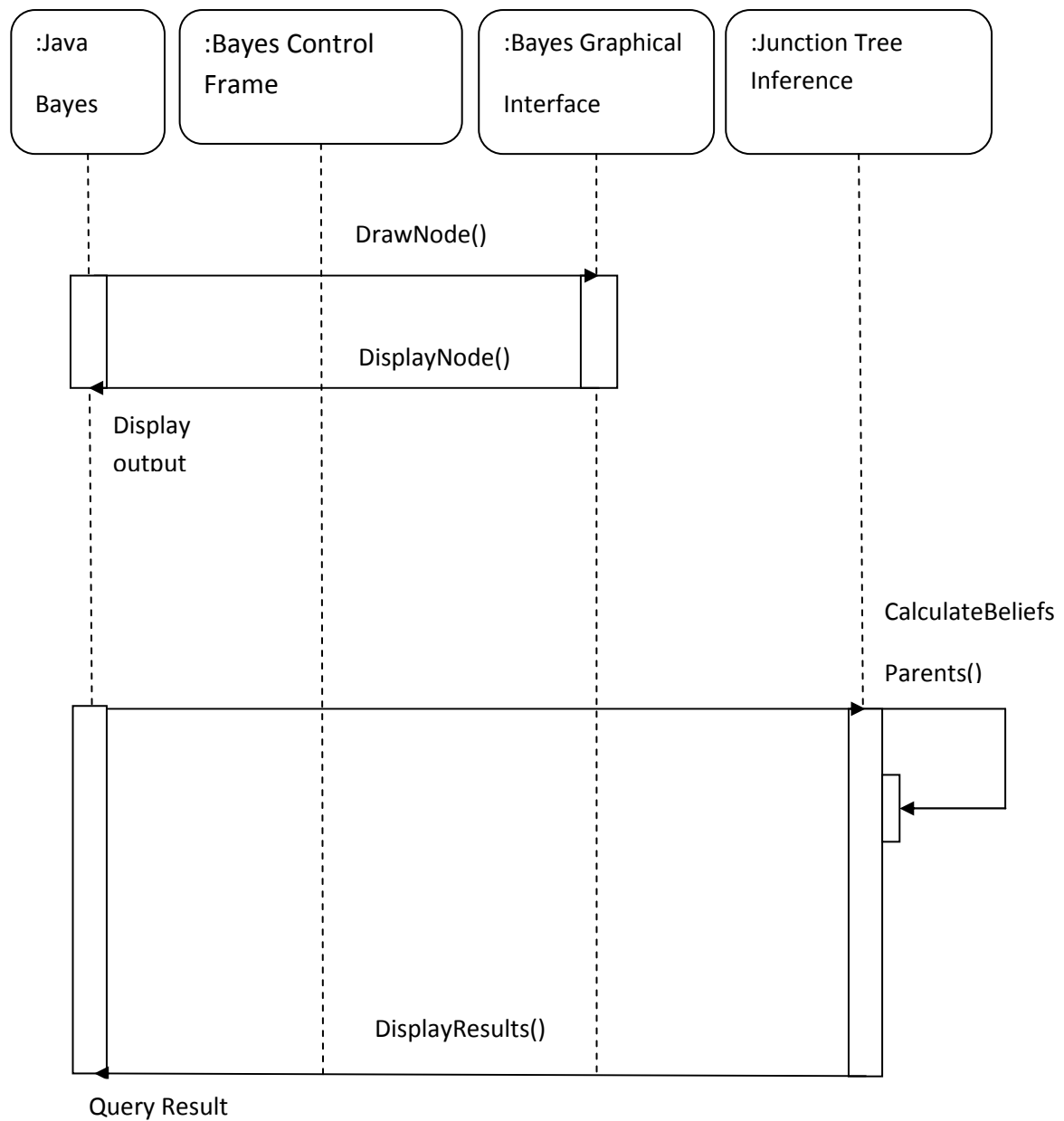


Fig 5.2: Sequence Diagram for JavaBayes; Creation and Querying nodes

5.3. Use Case Diagram

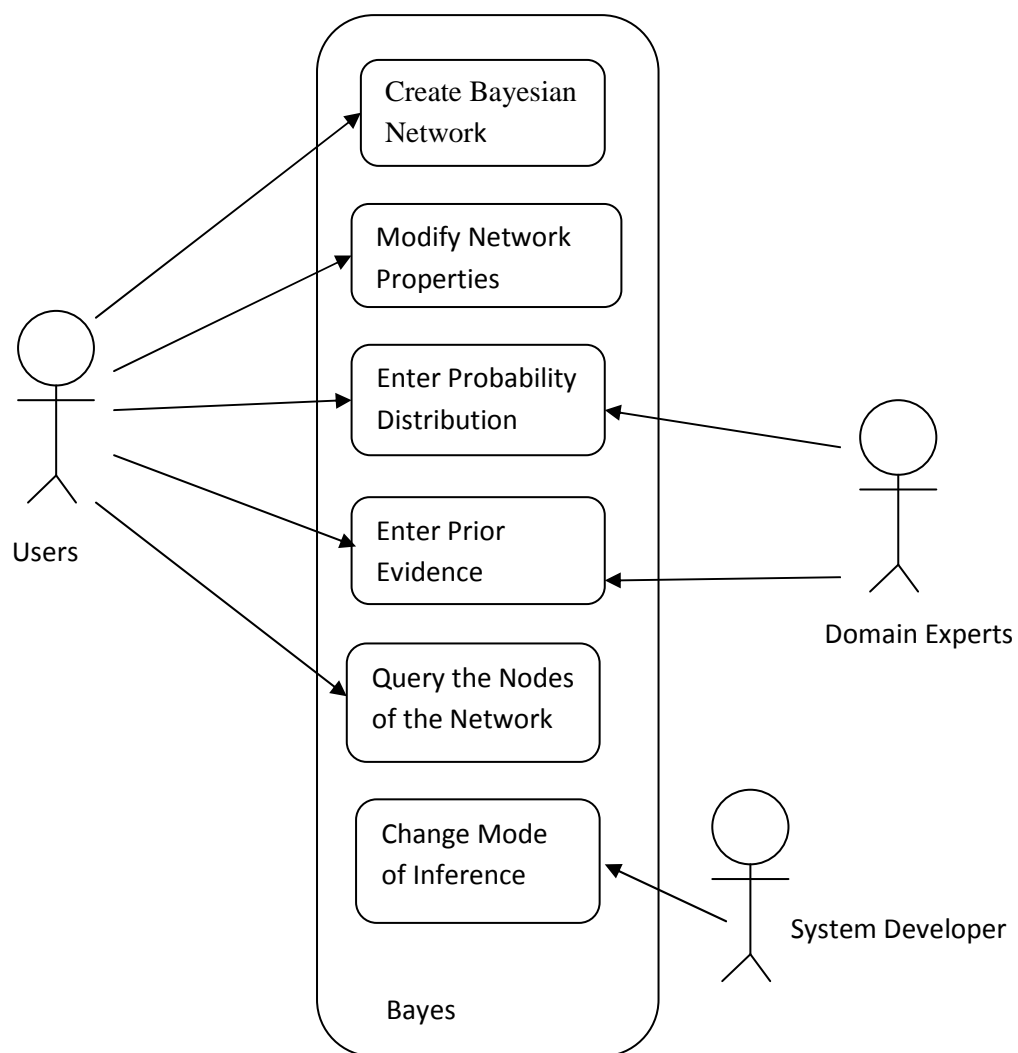


Fig 5.3: Use Case diagram

5.4. Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system. Data Flow models are used to show how data flows through a sequence of processing steps. The data is transformed at each step before moving on to the next stage. These processing steps or transformations are program functions when Data Flow diagrams are used to document a software design. DFD diagram is composed of four elements, which are process, data flow, external entity and data store.

5.4.1 Level 0 and Level 1 Data Flow Diagram

The level 0 is the initial level Data flow diagram and it's generally called as the context level diagram. It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then exploded to show more detail of the system being modeled.

Level 1 design diagram provides the details of the how the system is being divided into sub-systems (processes), where each processes deals with one or more data and control flows to or from an external agent, and which together provides each functionality of the system as a whole. Level 1 designing gives more sub division of each segment with various functions and internal operations, when compared to the top level 0 design. Level 1 design diagram flow provides the various control and data flow, here level 0 flow diagram is further sub divided into four level 1 flow diagram using UML notations.

CHAPTER 6

IMPLEMENTATION

6.1 Java Technology

Java technology is both a programming language and a platform.

6.2 The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

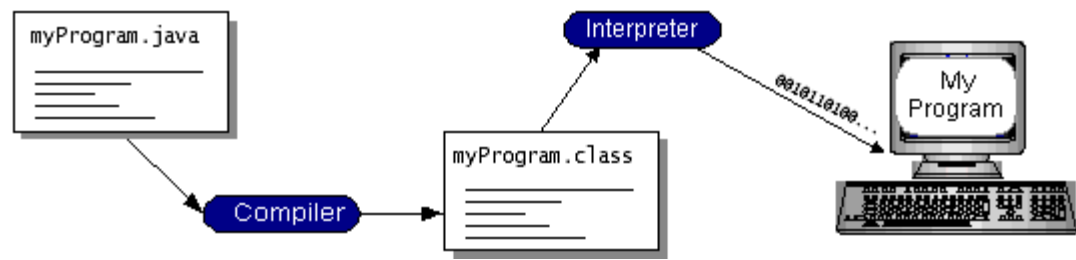


Fig 6.1 Java Virtual Machine

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

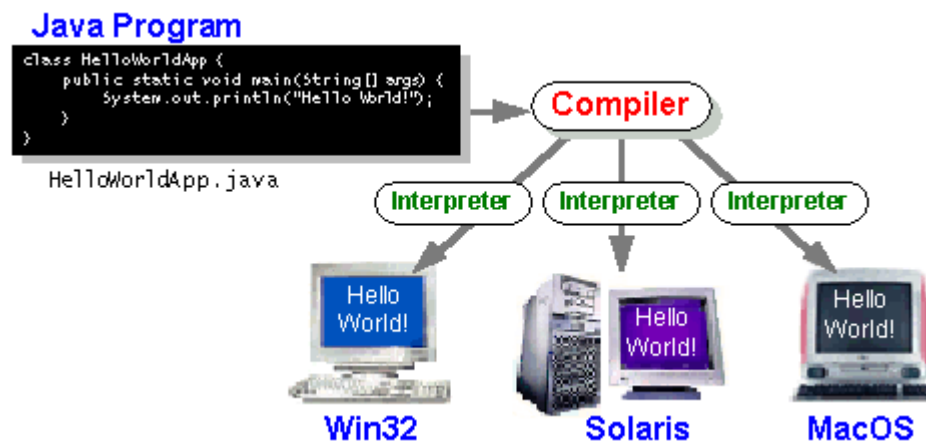


Fig 6.2 Java Compiler

6.3 The Java Platform

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The Java Virtual Machine (Java VM)

- The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.

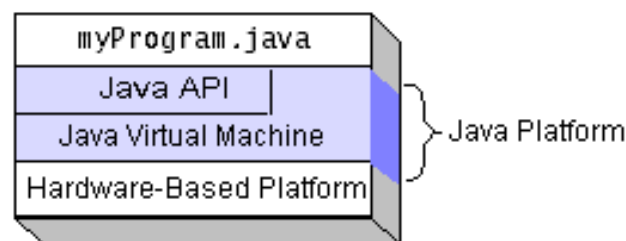


Fig 6.3 Running Java Platform

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

6.4 What Can Java Technology Do?

The most common types of programs written in the Java programming language are applets and applications. If you've browsed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlet are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The Essentials:** Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets:** The set of conventions used by applets.
- **Networking:** URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization:** Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security:** Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components:** Known as JavaBeansTM, can plug into existing component architectures.
- **Object serialization:** Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBCTM):** Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.

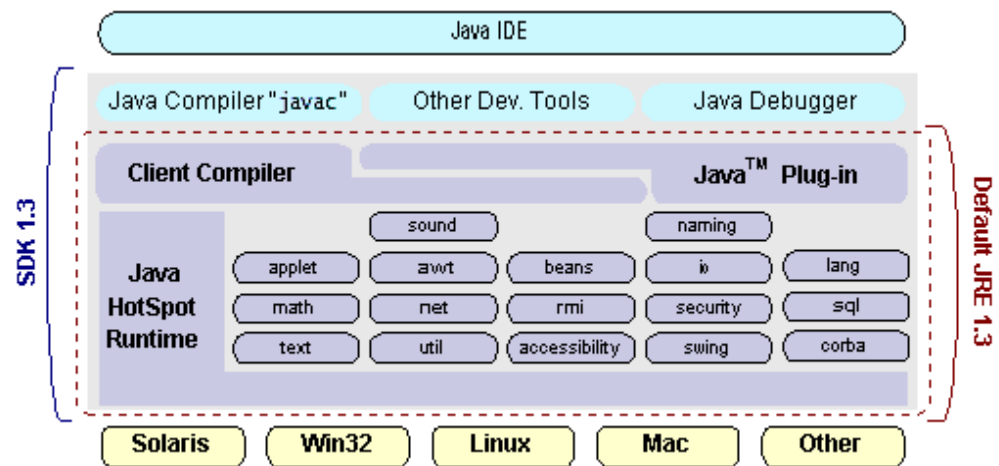


Fig 6.4 Java2 SDK

6.5. How Will Java Technology Change My Life?

We can't promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code:** The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java:** You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java™ Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
- **Write once, run anywhere:** Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.

- **Distribute software more easily:** You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

6.6 Class Diagram

The following is the class diagram for the project

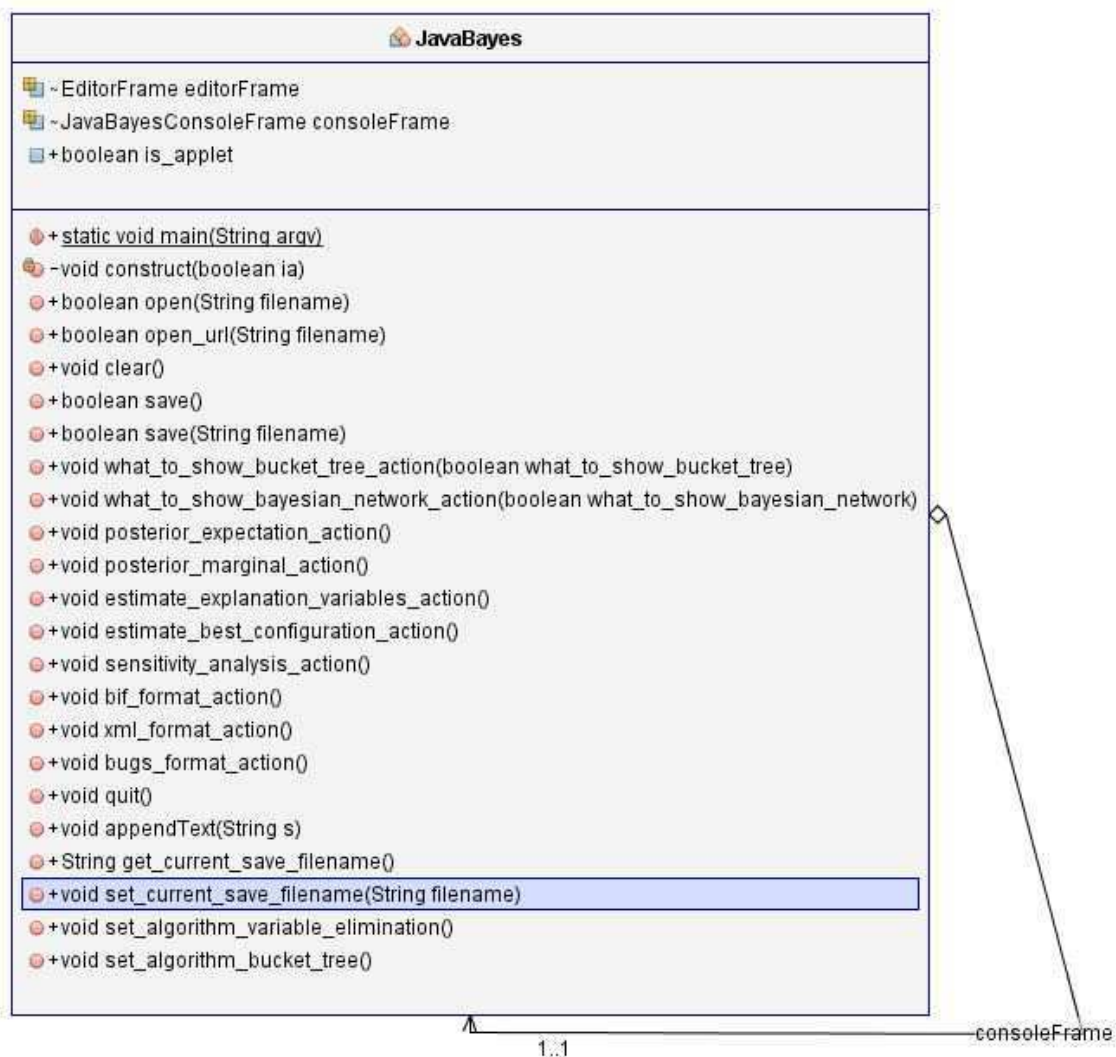


Fig 6.5: Class Diagram; shows the Class diagram for JavaBayes main program

CHAPTER 7

TESTING

Testing is an important phase in the development life cycle of the product; this is the phase where the error remaining from all the phases are detected. Hence testing performs a very critical role for quality assurance and ensuring the reliability of the software. During the testing, the program reliability of the software is tested. During the testing, the program to be tested is executed with a set of test cases and the output of the program for the test cases is evaluated to determine whether the program is performing as expected. Errors are found and corrected by using the following testing steps and correction is recorded for future references. Thus, a series of testing is performed on the system before it is ready for implementation.

7.1. Levels of Testing

7.1.1. Unit Testing

Unit testing focuses verification effort on the unit of software design (module). Using the unit test plans prepared in the design phase of the system development as a guide, important control paths are tested to uncover errors within the boundary of the modules. The interfaces of each of the module are tested to ensure proper flow of the information in and out of the modules under consideration. Boundary conditions are checked. All independent paths are exercised to ensure that all statements in the module are executed at least once and all error- handling paths are tested. Each unit is thoroughly tested to check if it might fall in any possible situation. This testing is carried out during the programming itself. At the end of this testing phase, each unit is found to be working satisfactorily, as regard to the expected output from the module.

7.1.2. Integration Testing

Data can be lost across an interface because one module can have an adverse effect on another's sub functions, when combined may not produce the desired major function, global data structures can present problems. Integration testing is a symmetric technique for the constructing the program structure while at the same time conducting tests to uncover errors associated with the interface. All modules are combined in this testing step. Then the entire program is tested as a whole.

7.1.3. System Testing

After the integration testing, the software is completely assembled as a package, interfacing errors have been uncovered and corrected and the final series of software validation tests begin. Validation test succeeds when the software functions in a manner that can be reasonably expected by the customer. Here the system is tested against system requirement specification. System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all works to verify that all system elements have been properly integrated and perform allocated functions.

7.2. Types of Testing

7.2.1 Acceptance Testing

In engineering and its various sub disciplines, acceptance testing is a test conducted to determine if the requirements of specification or contract are met. It may involve chemical tests, physical tests, or performance tests

In system engineering it may involve black-box testing performed on a system (for example: a piece of software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery. It is also known as functional testing, black-box testing, QA testing, application testing, confidence testing, final testing, validation testing, or factory acceptance testing. Software developers often distinguish acceptance testing by the system provider from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership. In the case of software, acceptance testing performed by the customer is known as user acceptance testing (UAT), end-user testing, site (acceptance) testing, or field (acceptance) testing.

7.2.2. Correctness testing

Correctness is the minimum requirement of software, the essential purpose of testing. Correctness testing will need some type of oracle, to tell the right behavior from the wrong one. The tester may or may not know the inside details of the software module under test, e.g. control flow, data flow, etc. Therefore, either a white-box point of view or black-box point of view can be taken in testing software. We must note that the black-box and white-box ideas are not limited in correctness testing only.

7.2.3. Black-box testing

The black-box approach is a testing method in which test data are derived from the specified functional requirements without regard to the final program structure. [Perry90] It is also termed data-driven, input/output driven [Myers79], or requirements-based [Hetzel88] testing. Because only the functionality of the software module is of concern, black-box testing also mainly refers to functional testing -- a testing method emphasized on executing the functions and examination of their input and output data. [Howden87] The tester treats the software under test as a black box -- only the inputs, outputs and specification are visible, and the functionality is determined by observing the outputs to corresponding inputs. In testing, various inputs are exercised and the outputs are compared against specification to validate the correctness. All test cases are derived from the specification. No implementation details of the code are considered.

It is obvious that the more we have covered in the input space, the more problems we will find and therefore we will be more confident about the quality of the software. Ideally we would be tempted to exhaustively test the input space. But as stated above, exhaustively testing the combinations of valid inputs will be impossible for most of the programs, let alone considering invalid inputs, timing, sequence, and resource variables. Combinatorial explosion is the major roadblock in functional testing. To make things worse, we can never be sure whether the specification is either correct or complete. Due to limitations of the language used in the specifications (usually natural language), ambiguity is often inevitable. Even if we use some type of formal or restricted language, we may still fail to write down all the possible cases in the specification. Sometimes, the specification itself becomes an intractable problem: it is not possible to specify precisely every situation that can be encountered using limited words. And people can seldom specify clearly what they want -- they usually can tell whether a prototype is, or is not, what they want after they have been finished. Specification problems contribute approximately 30 percent of all bugs in software. [Beizer95]

The research in black-box testing mainly focuses on how to maximize the effectiveness of testing with minimum cost, usually the number of test cases. It is not possible to exhaust the input space, but it is possible to exhaustively test a subset of the input space. Partitioning is one of the common techniques. If we have partitioned the input space and assume all the input values in a partition is equivalent, then we

only need to test one representative value in each partition to sufficiently cover the whole input space. Domain testing [Beizer95] partitions the input domain into regions, and considers the input values in each domain an equivalent class. Domains can be exhaustively tested and covered by selecting a representative value(s) in each domain. Boundary values are of special interest. Experience shows that test cases that explore boundary conditions have a higher payoff than test cases that do not. Boundary value analysis [Myers79] requires one or more boundary values selected as representative test cases. The difficulties with domain testing are that incorrect domain definitions in the specification cannot be efficiently discovered.

Good partitioning requires knowledge of the software structure. A good testing plan will not only contain black-box testing, but also white-box approaches, and combinations of the two.

7.2.4. White-box testing

Contrary to black-box testing, software is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester. Testing plans are made according to the details of the software implementation, such as programming language, logic, and styles. Test cases are derived from the program structure. White-box testing is also called glass-box testing, logic-driven testing [Myers79] or design-based testing [Hetzel88].

There are many techniques available in white-box testing, because the problem of intractability is eased by specific knowledge and attention on the structure of the software under test. The intention of exhausting some aspect of the software is still strong in white-box testing, and some degree of exhaustion can be achieved, such as executing each line of code at least once (statement coverage), traverse every branch statements (branch coverage), or cover all the possible combinations of true and false condition predicates (Multiple condition coverage). [Parrington89]

Control-flow testing, loop testing, and data-flow testing, all maps the corresponding flow structure of the software into a directed graph. Test cases are carefully selected based on the criterion that all the nodes or paths are covered or traversed at least once. By doing so we may discover unnecessary "dead" code -- code that is of no use, or never get executed at all, which cannot be discovered by functional testing.

In mutation testing, the original program code is perturbed and many mutated programs are created, each contains one fault. Each faulty version of the program is called a mutant. Test data are selected based on the effectiveness of failing the mutants. The more mutants a test case can kill, the better the test case is considered. The problem with mutation testing is that it is too computationally expensive to use. The boundary between black-box approach and white-box approach is not clear-cut. Many testing strategies mentioned above, may not be safely classified into black-box testing or white-box testing. It is also true for transaction-flow testing, syntax testing, finite-state testing, and many other testing strategies not discussed in this text. One reason is that all the above techniques will need some knowledge of the specification of the software under test. Another reason is that the idea of specification itself is broad -- it may contain any requirement including the structure, programming language, and programming style as part of the specification content.

We may be reluctant to consider random testing as a testing technique. The test case selection is simple and straightforward: they are randomly chosen. Study in [Duran84] indicates that random testing is more cost effective for many programs. Some very subtle errors can be discovered with low cost. And it is also not inferior in coverage than other carefully designed testing techniques. One can also obtain reliability estimate using random testing results based on operational profiles. Effectively combining random testing with other testing techniques may yield more powerful and cost-effective testing strategies.

7.2.5. Performance testing

Not all software systems have specifications on performance explicitly. But every system will have implicit performance requirements. The software should not take infinite time or infinite resource to execute. "Performance bugs" sometimes are used to refer to those design problems in software that cause the system performance to degrade.

Performance has always been a great concern and a driving force of computer evolution. Performance evaluation of a software system usually includes: resource usage, throughput, and stimulus-response time and queue lengths detailing the average or maximum number of tasks waiting to be serviced by selected resources. Typical resources that need to be considered include network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage

[Smith90]. The goal of performance testing can be performance bottleneck identification, performance comparison and evaluation, etc. The typical method of doing performance testing is using a benchmark -- a program, workload or trace designed to be representative of the typical system usage. [Vokolos98]

7.2.6. Reliability testing

Software reliability refers to the probability of failure-free operation of a system. It is related to many aspects of software, including the testing process. Directly estimating software reliability by quantifying its related factors can be difficult. Testing is an effective sampling method to measure software reliability. Guided by the operational profile, software testing (usually black-box testing) can be used to obtain failure data, and an estimation model can be further used to analyze the data to estimate the present reliability and predict future reliability. Therefore, based on the estimation, the developers can decide whether to release the software, and the users can decide whether to adopt and use the software. Risk of using software can also be assessed based on reliability information.

The robustness of a software component is the degree to which it can function correctly in the presence of exceptional inputs or stressful environmental conditions. Robustness testing differs with correctness testing in the sense that the functional correctness of the software is not of concern. It only watches for robustness problems such as machine crashes, process hangs or abnormal termination. The oracle is relatively simple; therefore robustness testing can be made more portable and scalable than correctness testing.

Stress testing, or load testing, is often used to test the whole system rather than the software alone. In such tests the software or system are exercised with or beyond the specified limits. Typical stress includes resource exhaustion, bursts of activities, and sustained high loads.

7.2.7. Security testing

Software quality, reliability and security are tightly coupled. Flaws in software can be exploited by intruders to open security holes. With the development of the Internet, software security problems are becoming even more severe.

Many critical software applications and services have integrated security measures against malicious attacks. The purpose of security testing of these systems

include identifying and removing software flaws that may potentially lead to security violations, and validating the effectiveness of security measures.

7.3. Unit Testing of Main Modules

7.3.1. Unit Testing of Creating Network

Test Case ID	Test 1
Test Description	Creating of Bayesian Network
Input	Creating nodes and connecting Edges.
Expected Output	To obtain the Directed Acyclic Graph.
Actual Output	To obtain the Directed Acyclic Graph.
Test Result	PASS

Table 7.1: Creating a network

Test Case ID	Test 2
Test Description	Enter the probabilistic values.
Input	Enter the range between 0 and 1.
Expected Output	Obtaining a valid value.
Actual Output	Obtaining a valid value.
Test Result	PASS

Table 7.2: Entering probabilistic values

7.3.2. Unit Testing of Entering Evidence

Test Case ID	Test 3
Test Description	Providing Evidence
Input	Entering true or false.
Expected Output	To check if the given node is observed or not.
Actual Output	To check if the given node is observed or not.
Test Result	PASS

Table 7.3: Setting Evidence Value

7.3.3. Unit Testing For Querying Nodes

Test Case ID	Test 4
Test Description	Querying a node
Input	Clicking on the particular node.
Expected Output	Obtain the probability value of the chosen node.
Actual Output	Obtain the probability value of the chosen node.
Test Result	PASS

Table 7.4: Querying a node

CHAPTER 8

RESULTS

Network Editor

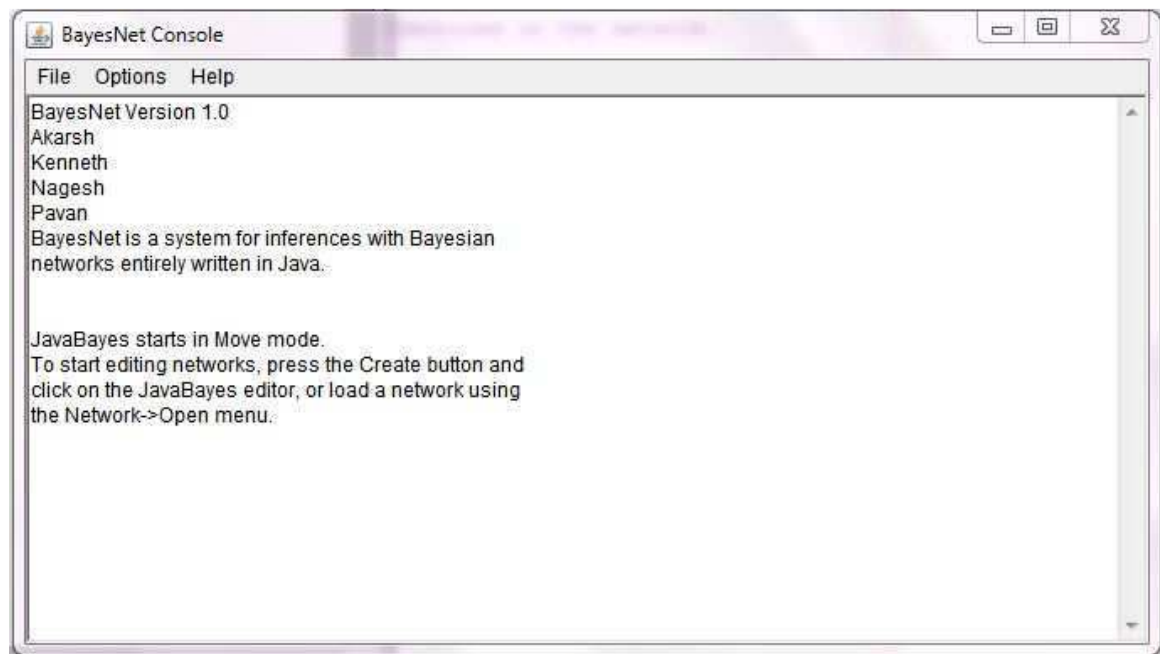


Fig 8.1: Console Frame

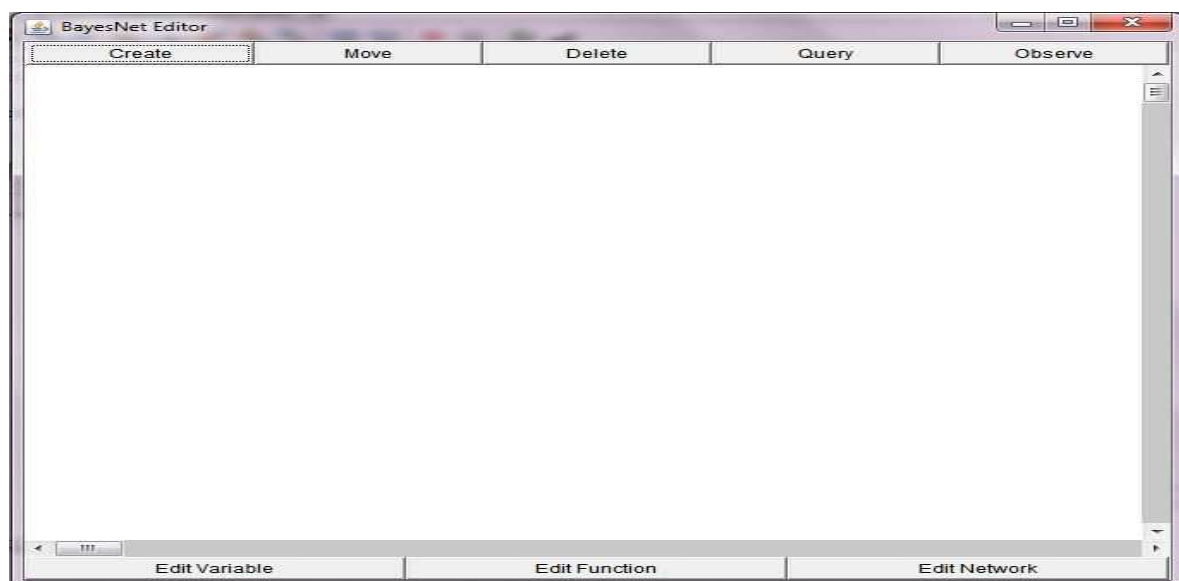


Fig 8.2: Editor Frame

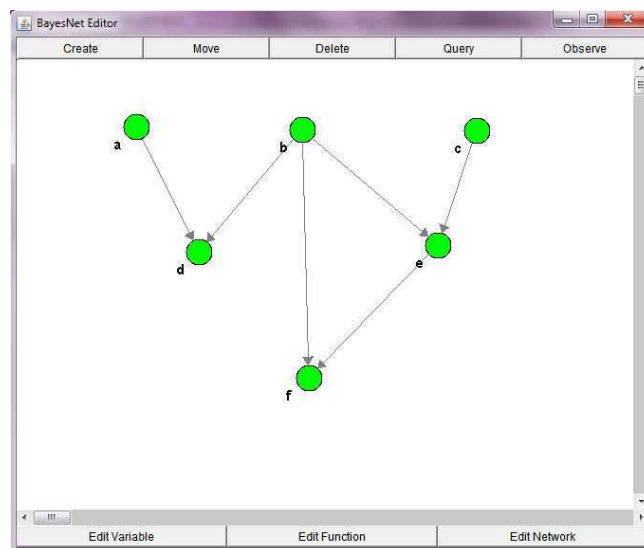


Fig 8.3: Creating a network

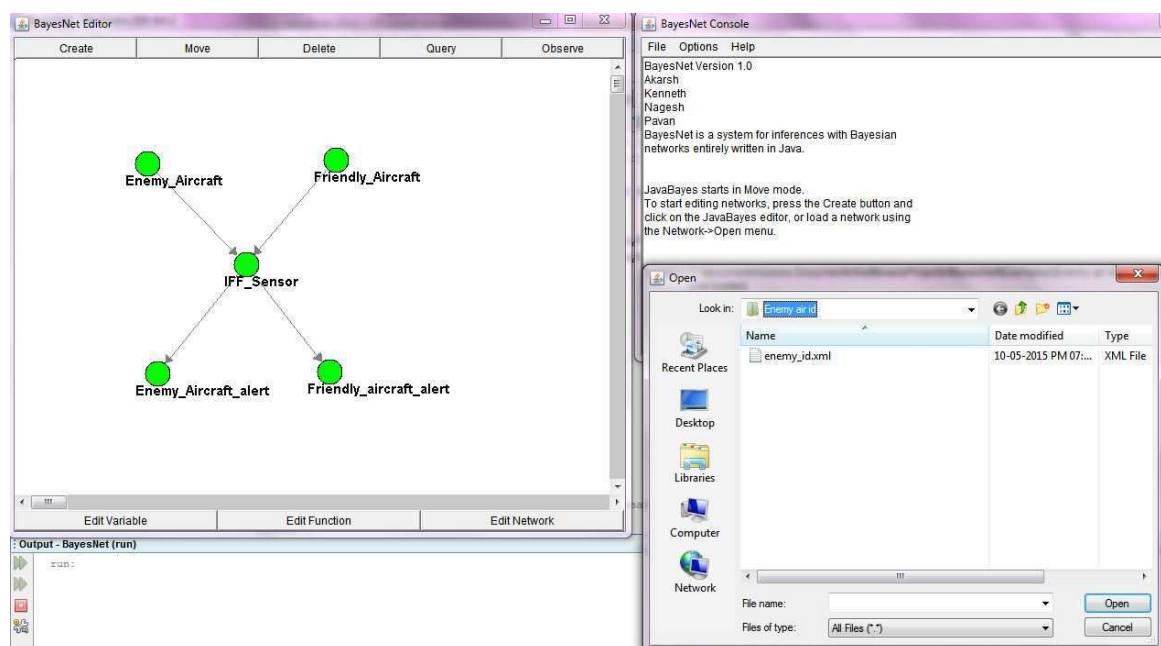


Fig 8.4: Opening a network file

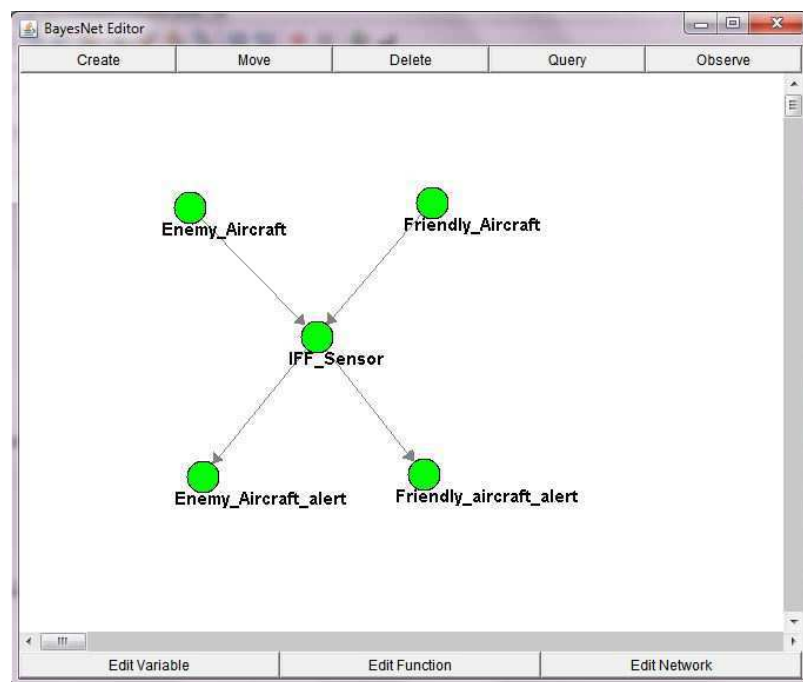


Fig 8.5: Enemy aircraft identifier network

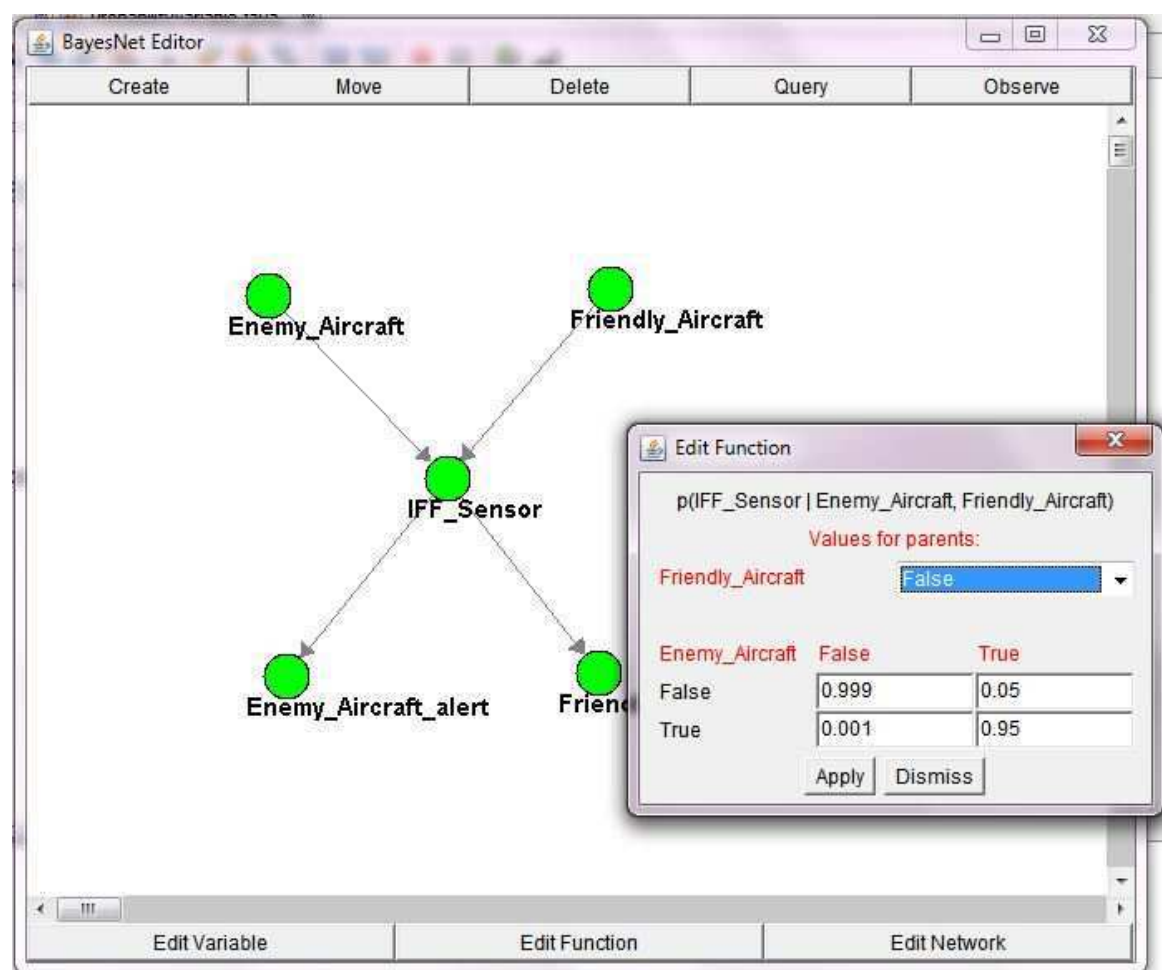


Fig 8.6: Entering Probability Distribution

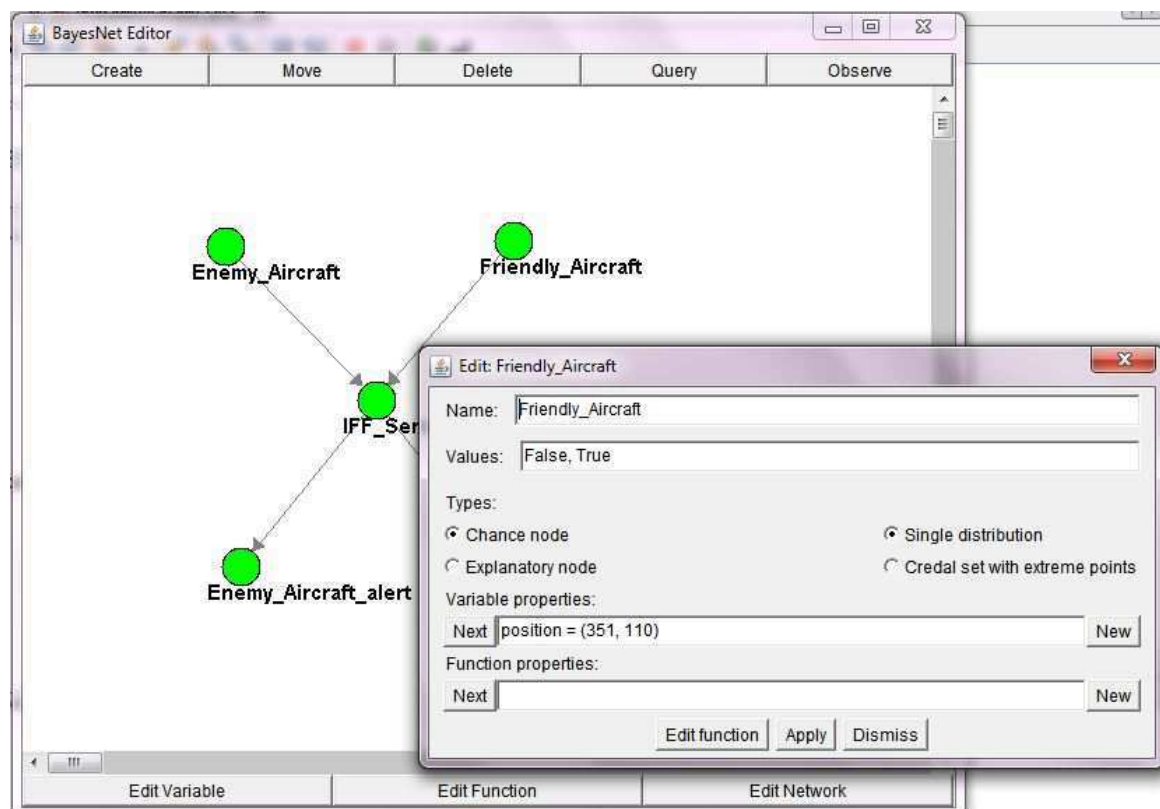


Fig 8.7: Editing network properties

Querying the Network

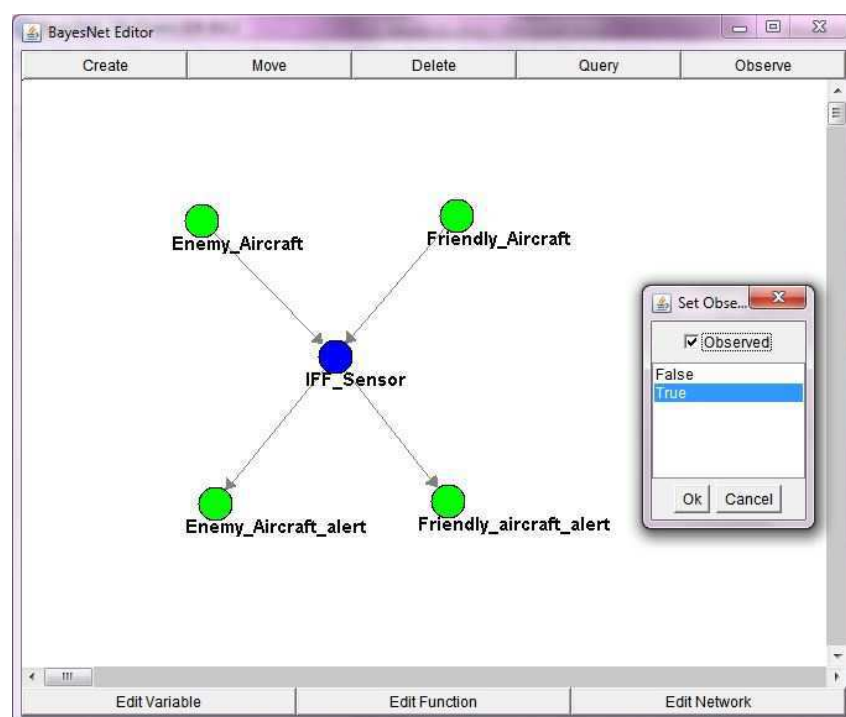


Fig 8.8: Providing Evidence

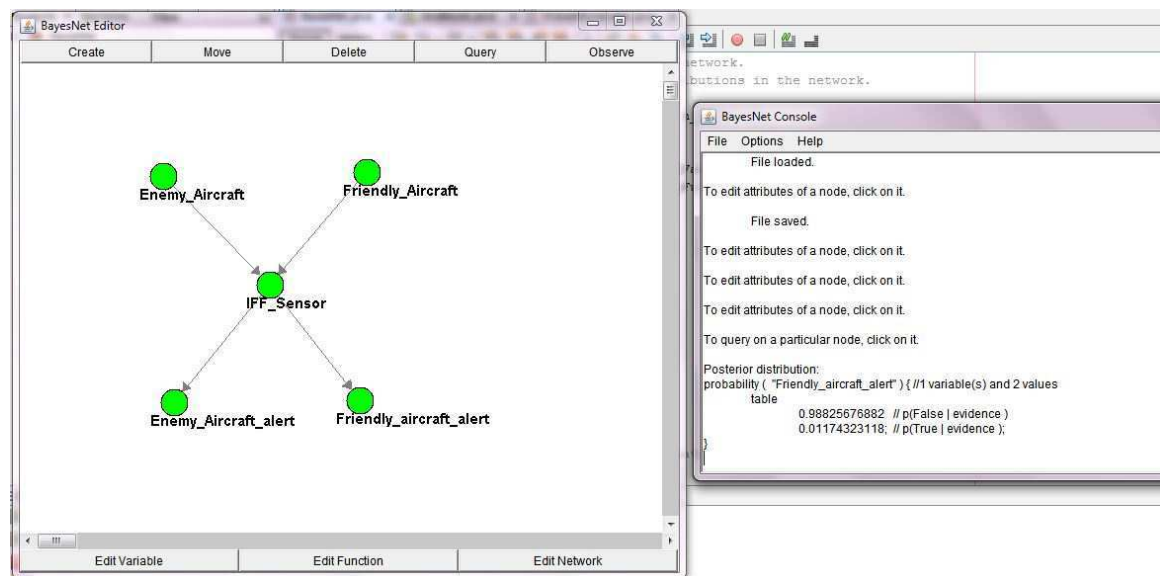


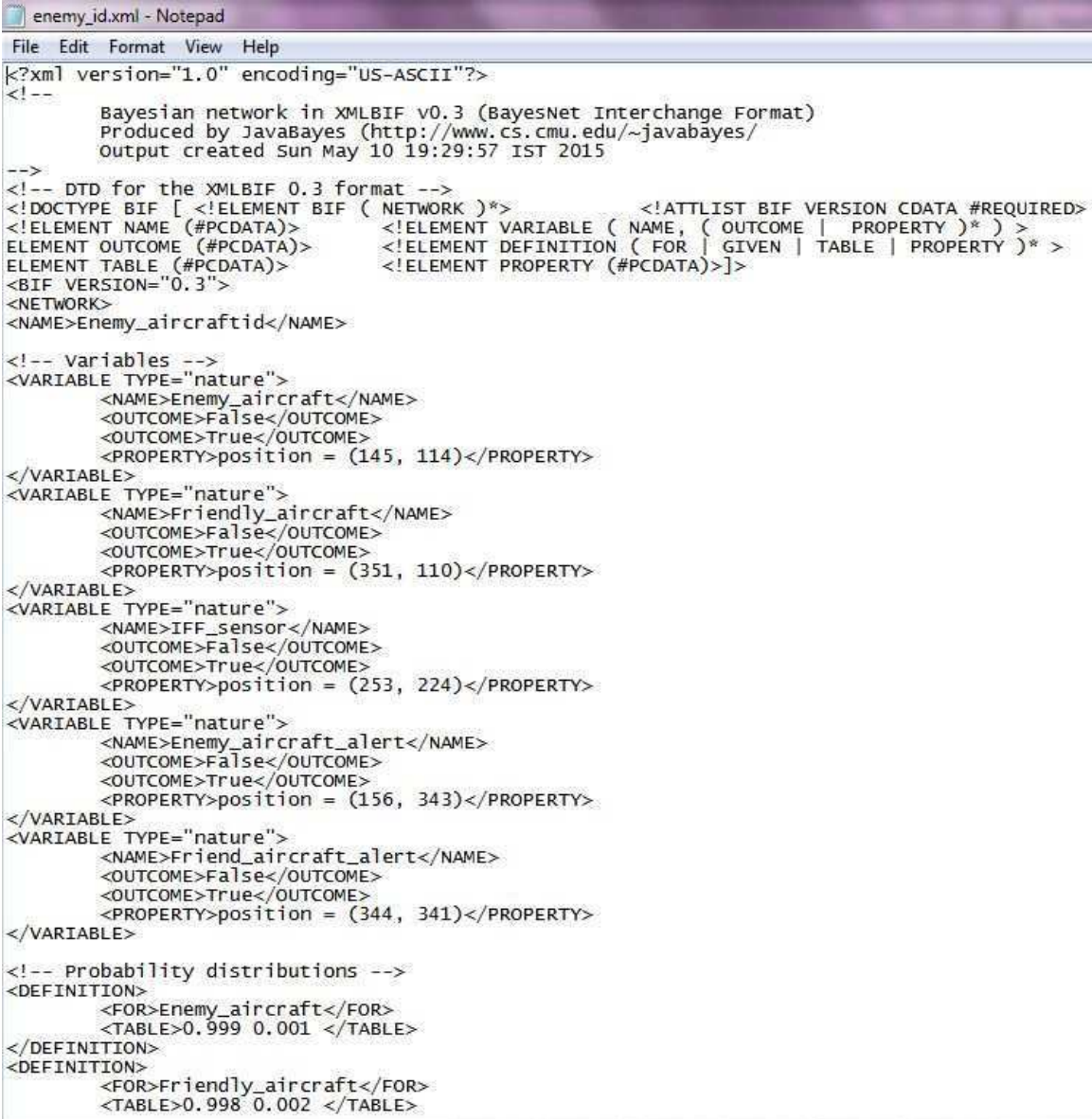
Fig8.9: Querying a Node

```

BayesNet Console
File Options Help
// Bayesian network
network "Enemy_aircraftid" { //5 variables and 5 probability distributions
}
variable "Enemy_aircraft" { //2 values
type discrete[2] { "False" "True" };
property "position = (145, 114)";
}
variable "Friendly_aircraft" { //2 values
type discrete[2] { "False" "True" };
property "position = (351, 110)";
}
variable "IFF_sensor" { //2 values
type discrete[2] { "False" "True" };
property "position = (253, 224)";
}
variable "Enemy_aircraft_alert" { //2 values
type discrete[2] { "False" "True" };
property "position = (156, 343)";
}
variable "Friend_aircraft_alert" { //2 values
type discrete[2] { "False" "True" };
property "position = (344, 341)";
}
probability ( "Enemy_aircraft" ) { //1 variable(s) and 2 values
table
0.999 // p(False | evidence )
0.001; // p(True | evidence );
}
probability ( "Friendly_aircraft" ) { //1 variable(s) and 2 values
table
0.998 // p(False | evidence )
0.002; // p(True | evidence );
}
probability ( "IFF_sensor" "Enemy_aircraft" "Friendly_aircraft" ) { //3 variable(s) and 8 values
table
0.999 0.71 0.05 0.05 0.001 0.29 0.95 0.95;
}
probability ( "Enemy_aircraft_alert" "IFF_sensor" ) { //2 variable(s) and 4 values
table
0.95 0.1 0.05 0.9;
}
probability ( "Friend_aircraft_alert" "IFF_sensor" ) { //2 variable(s) and 4 values
table
0.99 0.3 0.01 0.7;
}
Posterior distribution:
probability ( "IFF_sensor" ) { //1 variable(s) and 2 values

```

Fig 8.10: Bayesian network query



```

<?xml version="1.0" encoding="US-ASCII"?>
<!--
    Bayesian network in XMLBIF v0.3 (BayesNet Interchange Format)
    Produced by JavaBayes (http://www.cs.cmu.edu/~javabayes/)
    Output created Sun May 10 19:29:57 IST 2015
-->
<!-- DTD for the XMLBIF 0.3 format -->
<!DOCTYPE BIF [ <!ELEMENT BIF ( NETWORK )*>                <!ATTLIST BIF VERSION CDATA #REQUIRED>
<!ELEMENT NAME (#PCDATA)>                <!ELEMENT VARIABLE ( NAME, ( OUTCOME | PROPERTY )* ) >
<ELEMENT OUTCOME (#PCDATA)>                <!ELEMENT DEFINITION ( FOR | GIVEN | TABLE | PROPERTY )* >
<ELEMENT TABLE (#PCDATA)>                <!ELEMENT PROPERTY (#PCDATA)> ]>
<BIF VERSION="0.3">
<NETWORK>
<NAME>Enemy_aircraftid</NAME>

<!-- Variables -->
<VARIABLE TYPE="nature">
    <NAME>Enemy_aircraft</NAME>
    <OUTCOME>False</OUTCOME>
    <OUTCOME>True</OUTCOME>
    <PROPERTY>position = (145, 114)</PROPERTY>
</VARIABLE>
<VARIABLE TYPE="nature">
    <NAME>Friendly_aircraft</NAME>
    <OUTCOME>False</OUTCOME>
    <OUTCOME>True</OUTCOME>
    <PROPERTY>position = (351, 110)</PROPERTY>
</VARIABLE>
<VARIABLE TYPE="nature">
    <NAME>IFF_sensor</NAME>
    <OUTCOME>False</OUTCOME>
    <OUTCOME>True</OUTCOME>
    <PROPERTY>position = (253, 224)</PROPERTY>
</VARIABLE>
<VARIABLE TYPE="nature">
    <NAME>Enemy_aircraft_alert</NAME>
    <OUTCOME>False</OUTCOME>
    <OUTCOME>True</OUTCOME>
    <PROPERTY>position = (156, 343)</PROPERTY>
</VARIABLE>
<VARIABLE TYPE="nature">
    <NAME>Friend_aircraft_alert</NAME>
    <OUTCOME>False</OUTCOME>
    <OUTCOME>True</OUTCOME>
    <PROPERTY>position = (344, 341)</PROPERTY>
</VARIABLE>

<!-- Probability distributions -->
<DEFINITION>
    <FOR>Enemy_aircraft</FOR>
    <TABLE>0.999 0.001 </TABLE>
</DEFINITION>
<DEFINITION>
    <FOR>Friendly_aircraft</FOR>
    <TABLE>0.998 0.002 </TABLE>

```

Fig 8.11: Saved network in XML format

CHAPTER 9

SOURCE CODE

JavaBayes.java

```
package bayesnet;

/**
 *
 * @authors Akarsh, Kenneth, Nagesh, Pavan
 */
import JavaBayesInterface.*;
import InferenceGraphs.*;

public class JavaBayes{
    // Graphical elements of JavaBayes
    EditorFrame editorFrame;
    JavaBayesConsoleFrame consoleFrame;

    // Applet flags
    public boolean is_applet = false;

    /**
     * Main method for JavaBayes.
     */
    public static void main(String argv[]) {
        JavaBayes jb = new JavaBayes();
        jb.construct(false);
        if (argv.length > 0) {
            String filename = argv[0];
            System.out.println(filename);
            jb.open(filename);
        }
    }
}
```



```
/*
 * Do all the initializations for a JavaBayes object.
 */
private void construct(boolean ia) {
    is_applet = ia;

    editorFrame = new EditorFrame(this, "BayesNet Editor");
    editorFrame.show();
    consoleFrame = new JavaBayesConsoleFrame(this, "BayesNet Console");
    consoleFrame.show();

    JavaBayesHelpMessages.insert(this);
    JavaBayesHelpMessages.show(JavaBayesHelpMessages.about_message);
    JavaBayesHelpMessages.show(JavaBayesHelpMessages.start_message);
}

/**
 * Open a file and read the network in it.
 */
public boolean open(String filename) {
    return( editorFrame.open(filename) );
}

/**
 * Open a URL and read the network in it.
 */
public boolean open_url(String filename) {
    return( editorFrame.open_url(filename) );
}

/**
 * Clear the network.
 */
public void clear() {
    editorFrame.clear();
}
```

```
        set_current_save_filename(null);
    }

    /**
     * Save the network.
     */
    public boolean save() {
        return(editorFrame.save());
    }

    /**
     * Save the network.
     */
    public boolean save(String filename) {
        return(editorFrame.save(filename));
    }

    /**
     * Interact with menu options: whether to show BucketTree.
     */
    public void what_to_show_bucket_tree_action(boolean what_to_show_bucket_tree) {
        editorFrame.what_to_show_bucket_tree_action(what_to_show_bucket_tree);
    }

    /**
     * Interact with menu options: whether to show bayesian networks.
     */
    public void what_to_show_bayesian_network_action(boolean
what_to_show_bayesian_network) {

        editorFrame.what_to_show_bayesian_network_action(what_to_show_bayesian_network)
;
    }
```

```
/**
 * Inferences produce expectations.
 */
public void posterior_expectation_action() {
    editorFrame.posterior_expectation_action();
}

/**
 * Inferences produce posterior marginals.
 */
public void posterior_marginal_action() {
    editorFrame.posterior_marginal_action();
}

/**
 * Estimate explanation variables.
 */
public void estimate_explanation_variables_action() {
    editorFrame.estimate_explanation_variables_action();
}

/**
 * Produce the estimates for the best configuration.
 */
public void estimate_best_configuration_action() {
    editorFrame.estimate_best_configuration_action();
}

/**
 * Produce sensitivity analysis.
 */
public void sensitivity_analysis_action() {
    editorFrame.sensitivity_analysis_action();
}
```

```
/**
 * Use bif format for saving.
 */
public void bif_format_action() {
    editorFrame.set_save_format(EditorFrame.BIF_FORMAT);
}

/**
 * Use xml format for saving.
 */
public void xml_format_action() {
    editorFrame.set_save_format(EditorFrame.XML_FORMAT);
}

/**
 * Use bugs format for saving.
 */
public void bugs_format_action() {
    editorFrame.set_save_format(EditorFrame.BUGS_FORMAT);
}

/**
 * Quit gracefully.
 */
public void quit() {

    System.exit(0);
}

/**
 * Put text in the consoleFrame.
 */
public void appendText(String s) {
    consoleFrame.appendText(s);
}
```

```
/**
 * Get the current filename for saving.
 */
public String get_current_save_filename() {
    return( editorFrame.get_current_save_filename() );
}

/**
 * Set the current filename for saving.
 */
public void set_current_save_filename(String filename) {
    editorFrame.set_current_save_filename(filename);
}

/**
 * Set the inference algorithm as variable elimination.
 */
public void set_algorithm_variable_elimination() {

editorFrame.set_algorithm(EditorFrame.ALGORITHM_VARIABLE_ELIMINATION);
}

/**
 * Set the inference algorithm as bucket tree.
 */
public void set_algorithm_bucket_tree() {
    editorFrame.set_algorithm(EditorFrame.ALGORITHM_BUCKET_TREE);
}
}
```

JavaBayesControlFrame.java

```
package JavaBayesInterface;

import bayesnet.JavaBayes;

import java.io.*;
import java.applet.*;
import java.awt.*;

public class JavaBayesConsoleFrame extends Frame {
    private JavaBayes jb;

    // Declare controls
    FileDialog OpenFileDialog;
    FileDialog SaveFileDialog;
    TextArea textArea1;

    // Declare menus
    MenuBar mainMenuBar;
    Menu menu1, menu2, menu3, menu4, menu5, menu6, menu7;

    // Declare checkboxes
    CheckboxMenuItem show_buckets, show_bayes_net;
    CheckboxMenuItem algorithm_variable_elimination, algorithm_bucket_tree;
    CheckboxMenuItem bif_format, bugs_format, xml_format;
    CheckboxMenuItem posterior_marginal, posterior_expectation;
    CheckboxMenuItem explanation, best_configuration, sensitivity_analysis;

    static final String applet_invalid_operation = "This operation is not allowed in an
    applet!";

    // Labels
    static final String fileLabel = "File";
    static final String optionsLabel = "Options";
    static final String saveLabel = "Save format";
```

```
static final String bifSaveLabel = "BIF format";
static final String xmlSaveLabel = "XML format";
final static String bugsSaveLabel = "BUGS format";
final static String open_dialog_title = "Open";
final static String save_dialog_title = "Save";
final static String open_menuitem_title = "Open...";
final static String open_url_menuitem_title = "Open URL...";
    final static String save_menuitem_title = "Save";
    final static String save_as_menuitem_title = "Save as...";
    final static String clear_menuitem_title = "Clear";
    final static String dump_console_menuitem_title = "Dump console...";
    final static String quit_menuitem_title = "Quit";
    final static String show_bayes_net_title = "Bayesian network";
    final static String show_buckets_title = "Bucket tree";
    final static String algorithm_variable_elimination_title = "Variable elimination";
    final static String algorithm_bucket_tree_title = "Junction tree";
    final static String what_to_show_title = "What to show";
    final static String algorithm_title = "Algorithm";
    final static String inference_mode_title = "Inference mode";
    final static String posterior_marginal_title = "Posterior marginal";
    final static String posterior_expectation_title = "Posterior expectation";
    final static String explanation_title = "Estimate explanatory variables";
    final static String best_configuration_title = "Find complete explanation";
    final static String sensitivity_analysis_title = "Sensitivity analysis";
    final static String help_title = "Help";
    final static String about_title = "About";

    void BucketTree_Action() {
        jb.what_to_show_bucket_tree_action( show_buckets.getState() );
    }

    void BayesianNetwork_Action() {
        jb.what_to_show_bayesian_network_action( show_bayes_net.getState() );
    }
```

```
void PosteriorExpectation_Action() {
    CheckboxMenuItem active_item =
        update_checkbox_menu(menu2, posterior_expectation,
posterior_marginal);
    if ( active_item == posterior_expectation )
        jb.posterior_expectation_action();
    else
        jb.posterior_marginal_action();
}

void PosteriorMarginal_Action() {
    CheckboxMenuItem active_item =
        update_checkbox_menu(menu2, posterior_marginal,
posterior_expectation);
    if ( active_item == posterior_expectation )
        jb.posterior_expectation_action();
    else
        jb.posterior_marginal_action();
}

void EstimateBestConfiguration_Action() {
    CheckboxMenuItem active_item =
        update_checkbox_menu(menu2, best_configuration, posterior_marginal);
    if ( active_item == posterior_marginal )
        jb.posterior_marginal_action();
    else
        jb.estimate_best_configuration_action();
}

void SensitivityAnalysis_Action() {
    CheckboxMenuItem active_item =
        update_checkbox_menu(menu2, sensitivity_analysis, posterior_marginal);
    if ( active_item == posterior_marginal )
        jb.posterior_marginal_action();
    else
```



```
        jb.sensitivity_analysis_action();
    }

    void EstimateExplanationVariables_Action() {
        CheckboxMenuItem active_item =
            update_checkbox_menu(menu2, explanation, posterior_marginal);
        if ( active_item == posterior_marginal )
            jb.posterior_marginal_action();
        else
            jb.estimate_explanation_variables_action();
    }

    void BifFormat_Action() {
        CheckboxMenuItem active_item =
            update_checkbox_menu(menu5, bif_format, xml_format);
        if ( active_item == bif_format )
            jb.bif_format_action();
        else
            jb.xml_format_action();
    }

    void XmlFormat_Action() {
        CheckboxMenuItem active_item =
            update_checkbox_menu(menu5, xml_format, bif_format);
        if ( active_item == xml_format )
            jb.xml_format_action();
        else
            jb.bif_format_action();
    }

    void BugsFormat_Action() {
        CheckboxMenuItem active_item =
            update_checkbox_menu(menu5, bugs_format, bif_format);
        if ( active_item == bugs_format )
            jb.bugs_format_action();
    }
}
```

```
else
    jb.bif_format_action();
}

void AlgorithmVariableElimination_Action() {
    CheckboxMenuItem active_item =
        update_checkbox_menu(menu7, algorithm_variable_elimination,
algorithm_bucket_tree);
    if ( active_item == algorithm_variable_elimination )
        jb.set_algorithm_variable_elimination();
    else
        jb.set_algorithm_bucket_tree();
}

void AlgorithmBucketTree_Action() {
    CheckboxMenuItem active_item =
        update_checkbox_menu(menu7, algorithm_bucket_tree,
algorithm_variable_elimination);
    if ( active_item == algorithm_bucket_tree )
        jb.set_algorithm_bucket_tree();
    else
        jb.set_algorithm_variable_elimination();
}

void DumpConsoleToFile_Action() {
    if (jb.is_applet) {
        textArea1.setText(applet_invalid_operation);
        return;
    }
    SaveFileDialog.show();
    String filename = SaveFileDialog.getFile();
    if (filename == null) return;
    filename = SaveFileDialog.getDirectory() + filename;
    try {
        FileOutputStream fileout = new FileOutputStream(filename);
```

```
        PrintStream out = new PrintStream(fileout);
        String t = textArea1.getText();
        textArea1.setText("");
        out.print(t);
        out.close();
        fileout.close();
    } catch (IOException e) {
        appendText("Dump aborted: " + e + "\n");
        return;
    }
    appendText("\tConsole dumped.\n\n");
}

void Clear_Action() {
    (new ClearDialog(this, jb, "Clear the Bayesian network?", true)).show();
}

void Save_Action() {
    if (jb.is_applet) {
        appendText(applet_invalid_operation);
        return;
    }
    if (jb.get_current_save_filename() == null)
        SaveAs_Action();
    else {
        jb.save();
        appendText("\tFile saved.\n\n");
    }
}

void SaveAs_Action() {
    if (jb.is_applet) {
        appendText(applet_invalid_operation);
        return;
    }
}
```

```
        SaveFileDialog.show();
        String filename = SaveFileDialog.getFile();
        if (filename == null) return;
        filename = SaveFileDialog.getDirectory() + filename;
        if (jb.save(filename) == true)
            appendText("\tFile saved.\n\n");
        else
            appendText("\tFile not saved correctly.\n\n");
        jb.set_current_save_filename(filename);
    }

    void Open_Action() {
        if (jb.is_applet) {
            textArea1.append(applet_invalid_operation);
            return;
        }
        OpenFileDialog.show();
        String filename = OpenFileDialog.getFile();
        if (filename == null) return;
        filename = OpenFileDialog.getDirectory() + filename;
        if (jb.open(filename) == true)
            appendText("\tFile loaded.\n\n");
        else
            appendText("\tFile not loaded correctly.\n\n");
    }

    void Open_URL_Action() {
        (new OpenURLDialog(this, jb, "Insert URL of network", true)).show();
    }

    void Quit_Action() {
        (new QuitDialog(this, jb, "Quit JavaBayes?", false)).show();
    }

    void About_Action() {
```

```
JavaBayesHelpMessages.show(JavaBayesHelpMessages.about_message);
    }

    /*
     * End of Network Actions.
     */

/**
 * Constructor for JavaBayesConsoleFrame.
 */
    public JavaBayesConsoleFrame(JavaBayes java_bayes, String title) {
        jb = java_bayes;
        setTitle(title);

        // Initialize controls.
        OpenFileDialog = new java.awt.FileDialog(this, open_dialog_title,
FileDialog.LOAD);
        SaveFileDialog = new java.awt.FileDialog(this, save_dialog_title,
FileDialog.SAVE);
        textArea1 = new java.awt.TextArea();
        add("Center", textArea1);

        // Menus.
        mainMenuBar = new java.awt.MenuBar();

        menu1 = new java.awt.Menu(fileLabel);
        menu1.add(open_menuitem_title);
        //menu1.add(open_url_menuitem_title);
        menu1.add(save_menuitem_title);
        menu1.add(save_as_menuitem_title);
        menu1.add(clear_menuitem_title);
        //menu1.add(dump_console_menuitem_title);
        menu1.addSeparator();
        menu1.add(quit_menuitem_title);
        mainMenuBar.add(menu1);
```

```
menu4 = new Menu(optionsLabel);

menu6 = new Menu(what_to_show_title);
menu6.add( show_bayes_net = new
CheckboxMenuItem(show_bayes_net_title) );
//menu6.add( show_buckets = new
CheckboxMenuItem(show_buckets_title) );
menu4.add(menu6);

menu7 = new Menu(algorithm_title);
menu7.add( algorithm_variable_elimination = new
CheckboxMenuItem(algorithm_variable_elimination_title) );
menu7.add( algorithm_bucket_tree = new
CheckboxMenuItem(algorithm_bucket_tree_title) );
menu4.add(menu7);

menu2 = new Menu(inference_mode_title);
menu2.add( posterior_marginal = new
CheckboxMenuItem(posterior_marginal_title));
menu2.add( posterior_expectation = new
CheckboxMenuItem(posterior_expectation_title));
//menu2.add( explanation = new CheckboxMenuItem(explanation_title));
//menu2.add( best_configuration = new
CheckboxMenuItem(best_configuration_title));
// menu2.add( sensitivity_analysis = new
CheckboxMenuItem(sensitivity_analysis_title));
menu4.add(menu2);

menu5 = new Menu(saveLabel);
menu5.add( bif_format = new CheckboxMenuItem(bifSaveLabel));
menu5.add( xml_format = new CheckboxMenuItem(xmlSaveLabel));
//menu5.add( bugs_format = new CheckboxMenuItem(bugsSaveLabel));
menu4.add(menu5);
```

```
        mainMenuBar.add(menu4);

        menu3 = new java.awt.Menu(help_title);
        menu3.add(about_title);
        // The following try block was contributed
        // by Jason Townsend, Nov 12 2000.
    try {
        mainMenuBar.setHelpMenu(menu3);
    } catch (Exception e) {
        mainMenuBar.add(menu3);
    }

    setMenuBar(mainMenuBar);

    // Initialize the inference menu
    posterior_marginal.setState(true); // Simulate a true state.
    PosteriorMarginal_Action();

    // Initialize the save format menu
    xml_format.setState(true); // Simulate a true state.
    XmlFormat_Action();

    // Initialize the algorithm menu
    algorithm_variable_elimination.setState(true); // Simulate a true state.
    AlgorithmVariableElimination_Action();

    // Resize the frame.
    Toolkit t = Toolkit.getDefaultToolkit();
    Dimension d = t.getScreenSize();

    d.width = d.width / 2;
    d.height = d.height / 2;
    resize(d);
    }

/**
```

```
* Constructor for JavaBayesConsoleFrame.
*/

public JavaBayesConsoleFrame(JavaBayes jb) {
    this(jb, ((String)null));
}

/**
 * Override show() so that Console does not superimpose EditorFrame directly.
 */
public void show() {
    move(50, 50);
    super.show();
}

/**
 * Override action() to get events.
 */
public boolean action(Event event, Object arg) {
    if (event.target instanceof MenuItem) {
        String label = (String) ( ((MenuItem)event.target).getLabel());
        if (label.equals(show_buckets_title)) {
            BucketTree_Action();
            return true;
        } else
        if (label.equals(show_bayes_net_title)) {
            BayesianNetwork_Action();
            return true;
        } else
        if (label.equals(posterior_expectation_title)) {
            PosteriorExpectation_Action();
            return true;
        } else
        if (label.equals(posterior_marginal_title)) {
            PosteriorMarginal_Action();
            return true;
        }
    }
}
```



```
        } else
        if (label.equals(best_configuration_title)) {
            EstimateBestConfiguration_Action();
            return true;
        } else
        if (label.equals(sensitivity_analysis_title)) {
            SensitivityAnalysis_Action();
            return true;
        } else
        if (label.equals(explanation_title)) {
            EstimateExplanationVariables_Action();
            return true;
        } else
        if (label.equals(bifSaveLabel)) {
            BifFormat_Action();
            return true;
        } else
        if (label.equals(xmlSaveLabel)) {
            XmlFormat_Action();
            return true;
        } else
        if (label.equals(bugsSaveLabel)) {
            BugsFormat_Action();
            return true;
        } else
        if (label.equals(algorithm_variable_elimination_title)) {
            AlgorithmVariableElimination_Action();
            return true;
        } else
        if (label.equals(algorithm_bucket_tree_title)) {
            AlgorithmBucketTree_Action();
            return true;
        } else
        if (label.equals(clear_menuitem_title)) {
            Clear_Action();
```

```
        return true;
    } else
    if (label.equals(dump_console_menuitem_title)) {
        DumpConsoleToFile_Action();
    } else
    if (label.equals(save_menuitem_title)) {
        Save_Action();
        return true;
    } else
    if (label.equals(save_as_menuitem_title)) {
        SaveAs_Action();
        return true;
    } else
    if (label.equals(open_menuitem_title)) {
        Open_Action();
        return true;
    }
    if (label.equals(open_url_menuitem_title)) {
        Open_URL_Action();
        return true;
    } else
    if (label.equals(quit_menuitem_title)) {
        Quit_Action();
        return true;
    } else
    if (label.equals(about_title)) {
        About_Action();
        return true;
    }
    }
    return super.action(event, arg);
}
```

```
/**
```

```
 * Handle the possible destruction of the window.
```

```
*/  
  
public boolean handleEvent(Event evt) {  
    if (evt.id == Event.WINDOW_DESTROY)  
        if (jb != null)  
            (new QuitDialog(this, jb, "Quit JavaBayes?", false)).show();  
    return(super.handleEvent(evt));  
}  
  
/**  
 * Place text in the text area.  
 */  
  
public void appendText(String text) {  
    textArea1.appendText(text);  
}  
  
/*  
 * Create the "radiobutton" behavior for the checkbox menu items.  
 * It returns the checkbox that got on.  
 */  
  
private CheckboxMenuItem update_checkbox_menu(Menu m, CheckboxMenuItem  
cur, CheckboxMenuItem def) {  
    boolean s = cur.getState();  
  
    if (s == false) { // If cur was on, then cur is still off and def is on.  
        def.setState(true);  
        return(def);  
    }  
    else { // If cur was off, then cur is on and all others are off.  
        for (int i=0; i<m.countItems(); i++) // Set all menu items to off,  
            ((CheckboxMenuItem)(m.getItem(i))).setState(false);  
        cur.setState(true); // then set cur back to on.  
        return(cur);  
    }  
}  
}
```

CHAPTER 10

CONCLUSION

Bayesian Networks have incredible power to offer assistance in a wide range of endeavors. They support the use of probabilistic inference to update and revise belief values.

Bayesian networks readily permit qualitative inferences without the computational inefficiencies of traditional joint probability determinations. In doing so, they support complex inference modeling including rational decision making systems, value of information and sensitivity analysis. As such, they are useful for causality analysis and through statistical induction they support a form of automated learning.

One of the limitations of Bayesian Networks is reducing the computational complexity of inference in Bayesian Networks. Current algorithms for inference convert a Bayesian Network to a junction tree structure given by a tree of clusters and the resulting complexity is time exponential in the size of a cluster.

Our data set takes only discrete values as input for the nodes in the Bayesian Network. Further work is needed on factorizing conditional expressions whose conditionals contain a mixture of Boolean, labeled and continuous nodes. Hence future work will involve construction of probability distribution table for Dynamic Bayesian Networks.

CHAPTER 11

REFERENCES

- [1] Hsu-Yung Cheng, Chih-Chia Weng, and Yi-Ying Chen, "Vehicle detection in aerial surveillance using dynamic bayesian networks," Image Processing, IEEE Transactions on, vol. 21, no. 4, pp. 2152-2159, April 2012.
- [2] S. J. Russell and P. Norvig, Artificial intelligence: A modern approach, 2nd ed., ser. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.
- [3] Bennacer, Self-Diagnosis Technique for Virtual Private Networks Combining Bayesian Networks and Case-Based Reasoning, January 2015.
- [4] Korb and Nicholson, "Bayesian Artificial Intelligence", Second Edition, 2011.
- [5] N. Bencomo, A. Belaggoun, and V. Issarny, "Bayesian artificial intelligence for tackling uncertainty in self-adaptive systems: the case of dynamic decision networks," in 2nd International NSF sponsored Workshop on Realizing Artificial Intelligence Synergies in Software Engineering RAISE, 2013.
- [6] Ceccons, ",Exploring Early Glaucoma and the Visual Field Test: Classification and Clustering Using Bayesian Networks", vol. 110, no. 6, pp.812 -819 1992,2014.
- [7] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference. Morgan Kaufmann Publishers, 1988.
- [8] D.J. Spiegelhalter and R.J. Cowell, "Learning in Probabilistic Expert Systems," Bayesian Statistics, vol. 4, pp. 447-465, 1992.
- [9] M. Neil, N.E. Fenton, S. Forey, and R. Harris, "Using Bayesian Belief Networks to Predict the Reliability of Military Vehicles," Computing and Control Eng. J., vol. 12, no. 1, pp. 11-20, 2001.
- [10] M. Neil, M. Tailor, D. Marquez, N. Fenton, and P. Hearty, "Modelling Dependable Systems Using Hybrid Bayesian Networks, "Reliability Eng. and System Safety, vol. 93, no. 7, pp. 933-939, <http://dx.doi.org/10.1016/j.ress.2007.03.009>, July 2008.
- [11] M. Neil, N. Fenton, and M. Tailor, "Using Bayesian Networks to Model Expected and Unexpected Operational Losses," Risk Analysis J., vol. 25, pp. 963-972, Aug. 2005.
- [12] R.J. Cowell, R.J. Verral, and Y.K. Yoon, "Modelling Operational Risk with Bayesian Networks," The J. Risk and Insurance, vol. 74, no. 4, pp 795-827, 2006.
- [13] F.V. Jensen, An Introduction to Bayesian Networks. UCL Press, 1996.
- [14] S.L. Lauritzen and D.J. Spiegelhalter, "Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems (with Discussion)," J. Royal Statistical Soc. Series B, vol. 50, no 2, pp. 157-224, 1988.

- [15] www.agenarisk.com, 2010.
- [16] M. Neil, N.M. Nielson, F. Fenton, and X. Nielson, "Building Large- Scale Bayesian Networks," *The Knowledge Eng. Rev.*, vol. 15, no. 3, pp. 257-284, 2000.
- [17] P. Shenoy and G. Shafer, "Propagating Belief Functions with Local Computations," *IEEE Expert*, vol. E-1, no. 3, pp. 43-52, Sept. 1986.
- [18] F. Jensen, S.L. Lauritzen, and K. Olesen, "Bayesian Updating in Recursive Graphical Models by Local Computations," *Computational Statistics Quarterly*, vol. 4, pp. 260-282, 1990.
- [19] M. Neil, M. Tailor, and D. Marquez, "Inference in Bayesian Networks Using Dynamic Discretization," *Statistics and Computing*, vol. 17, no. 3, pp. 219-233, Sept. 2007.
- [20] www.bayesianinference.com, 2009.
- [21] Peter Hearty, Norman Fenton, David Marquez, and Martin Neil. Predicting Project Velocity in XP Using a Learning Dynamic Bayesian Network Model, *IEEE Transactions on Software Engineering*, 2009, 35(1):124-137.
- [22] Pearl J. Fusion, "Propagation and structuring in belief networks[J]. *Artificial Intelligence*", 1986, 29 (3): 241-288.