ABSTRACT

AUTOMATIC CATEGORIZATION OF ABTRACTS THROUGH BAYESIAN NETWORKS

by
William Ramirez

This thesis presents a method for assigning abstracts of Artificial Intelligence papers to their area of the field. The technique is implemented by the use of a Bayesian network where relevant keywords extracted from the abstract being categorized, are entered as evidence and inferencing is made to determine potential subject areas. The structure of the Bayesian network represents the causal relationship between Artificial Intelligence keywords and subject areas. Keyword components of the network are selected from pre-categorized abstracts. The work reported here is part of a larger project to automatically assign papers to reviewers for Artificial Intelligence conferences. The process of assigning papers to reviewers begins by using the inference system reported here to derive Artificial Intelligence subject areas for such papers. Based on those subjects, another module can select reviewers according to their specialization and limited by conflicts of interest.

AUTOMATIC CATEGORIZATION OF ABSTRACTS THROUGH BAYESIAN NETWORKS

by
William Ramirez

A Master's Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 2000

APPROVAL PAGE

AUTOMATIC CATEGORIZATION OF ABSTRACTS THROUGH BAYESIAN NETWORKS

William Ramirez

Dr. Richard Scherl, Thesis Advisor
Professor of Computer and Information Science, NJIT

Date

Dr. James Geller, Committee Member.
Professor of Computer and Information Science, NJIT

Date

Dr. Yehoshua Perl, Committee Member.
Professor of Computer and Information Science, NJIT

Date

BIOGRAPHICAL SKETCH

Author : William Ramirez

Degree : Master of Science in Computer Science

Date: January 2000

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2000

- Bachelor of Science in Systems Engineering,
  Antioquia University, Medellin, Colombia, 1996

Major: Computer Science

To my beloved family for all their support

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Dr. Richard Scherl, who was my Thesis Adviser and the main source of information by providing personal books and the software that make this thesis a reality, and Dr. James Geller, who was co-directing the Reviewer Selection Project and the connection point source allowing me to link my work to the rest of the project.

I would also like to give special mention to a team member which whom I had some discussions and brought brilliant ideas to this thesis, Fredrik Ledin.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

In 1997, Professor James Geller, wrote a challenge paper for the International Joint Conference on Artificial Intelligence(IJCAI). The challenge problem was to automatically assign reviewers to papers for this conference. The same technology could be used for other conferences as well.

The paper "Challenge: How IJCAI 1999 can prove the value of AI by using AI" based its motivation of automatic assignment of reviewers in the way the current program for assigning papers to reviewers worked. The program used in 1997 for such a conference was basically a very limited program in many different manners. Created by Ramesh Patil at USC/ISI, this program was not even intended for this conference, but for the National Conference on Artificial Intelligence(AAAI), so extra processing was required to accommodate the program to the particular requirements of IJCAI. The general algorithm of this program was, that an author submitted a paper with a list of content areas that the paper could be categorized, this list could not be extended since it actually represented all the content areas included in Patil's program, after this information was input into the system, the program assigns papers to reviewers according to the content areas of expertise of them.

From the previous sentence, one can infer two basic problems : The partial analysis of the document from the author; the program was not intelligent enough to deduce subject areas, and the limitation of the program on accepting new content areas. In other words, the author describes this issue as the contradiction of the use of a non-artificial intelligence program for the Artificial Intelligence Community.

The proposal of Prof. Geller with regards to what could be improved, can be summarized in the following way : Authors will supply a list of Content areas from any topic of Artificial Intelligence, and an intelligent program based on these contents and by analysis of the body of the document will determine potential content areas and at the same time will find reviewers for those papers. In those cases where program results were not reliable, a data entry process will be required. Adding to this statement, this thesis aims to eliminate the need for the authors to submit content areas, since the system will generate them automatically.

Then, a general framework was developed in order for, the different teams that wanted to participate on this challenge, to have the same conditions. The framework consisted of electronic submission of paper for IJCAI 1997, so that all teams would have them available online. Authors were asked to submit a list of five to ten individuals that they would consider good reviewers for their papers. The process of selection of reviewers is conditioned as follows : Reviewers can not have the same affiliation as the author, author can not be presented as a suggested reviewer, author's doctoral advisor can not be a reviewer, people with whom the author has recently written cannot be potential reviewers, no reviewer can have more than thirty papers; papers must have a reviewer.

The World Wide Web was proposed as a source of information to fulfill the reviewers conditions.

A general processing example of how to solve the problem will give a better understanding on how this issue can be faced : abstract can be submitted online or in batch mode and a data entry system will process them. The knowledge based system contains information such as the content areas, potential reviewers and references. A keyword extractor program will draw out significant keywords from the abstracts and a categorizer program will analyze those keywords and make inferences about the potential areas of the paper. The last stage involves the assignment of reviewers to the potential areas of the paper. Reviewers are chosen according to their specialization and by matching certain restriction rules such as : no reviewers can be selected with the same affiliation as the author, the author of the paper can not be selected as a reviewer, the author's advisor is not accepted as a reviewer, a reviewer can not be assigned fifty papers, nor can a paper be reviewed by 50 persons, reviewers that the author has recently written a paper with are not allowed to be reviewers for such a paper. A reviewer selection module supports these constraints and returns as the output, appropriate reviewers based on these limitations. The system then, send data about reviewers previously entered in the system and potential content areas to a checking reviewer module which will verify reviewer's assignment conditions and determine appropriate reviewers for such a paper. Should there be no information pertinent to the reviewer, a web-crawler module would interface with World Wide Web Search Engines to try to find information about reviewers.

In answer to the challenge, a group of NJIT students started the development of what is called the "REVIEWER SELECTION" project. The general structure of the project consists of four different modules that communicate to each other : Data Entry module, composed of a web interface and a batch process, Data Analysis module, conformed by a keyword extractor program and keyword – abstract counter program that will generate valuable information to the rest of the system, the Inference module composed of an statistical program for keywords-areas and the Categorizer program that will infer potential content areas, and the last module is the Reviewer selection module which will determine the best reviewers according to the conditions described before in this chapter. The functional schema of the system is presented in .

In this figure all the different system components that interact, are displayed :

- **Web Interface:** It will be used as the online input of abstracts. So authors of papers for IJCAI, can submit papers by using this interface. Not only abstracts are entered, but also, the content areas given by the author and a list of five to ten potential reviewers chosen by the writer as well.
- **Batch Processing:** This batch processing will allow the processing of more than one abstract. These abstracts are submitted to the system by using a flat file. This module is required in order to gather paper from old conferences.
- **Keyword Extractor :** It will select significant words from the abstract in order to determine potential content areas. These words are called keywords and the program by using grammar rules will identify and then draw out verbs and nouns. Common nouns and verbs from the English glossary will be eliminated.

- **Data Analyzer :** This program generates statistical information that will be used for the inference engine. It takes as input the keywords extracted from the Keyword Extractor and try to find the number of occurrences of each keyword in the abstract being processed. As a result the Data analyzer will save information in the knowledge base about : keyword, paper, content area, and number of times the keyword occurs.
- **Statistics Program :** This process, based in the information gathered by the data analyzer, tries to make up some probabilistic values that would represent relationship between content areas and keywords. The result of this will be a table showing a probabilistic value representing the relationship between keyword and content area.
- **Inference Engine or Categorizer :** By the use of Bayesian networks, content of abstract can be analyzed and determined AI subject areas where it can be classified. Once the category of the paper is determined, reviewers may be assigned as they are in the Database as experts in that area. This is what this work is based on.
- **Reviewer Selection Program :** This program tries to find reviewers according to the content areas of a document. It is also responsible for resolving conflicts of interest about a particular reviewer (Author and reviewer belong to the same affiliation, Reviewer is also author of the paper, Reviewer has recently written a different document with this author, etc.). As a result, this will return a list of n reviewers, where n is a predefined value, with no conflicts of interest at all.

This thesis describe my work on the problem of automatically assigning categories to papers by using a Bayesian networks. The document is organized into : a theoretical framework about Bayesian networks, a discussion of how Bayesian networks can be used to perform the categorization problem, the implementation and Conclusions. The

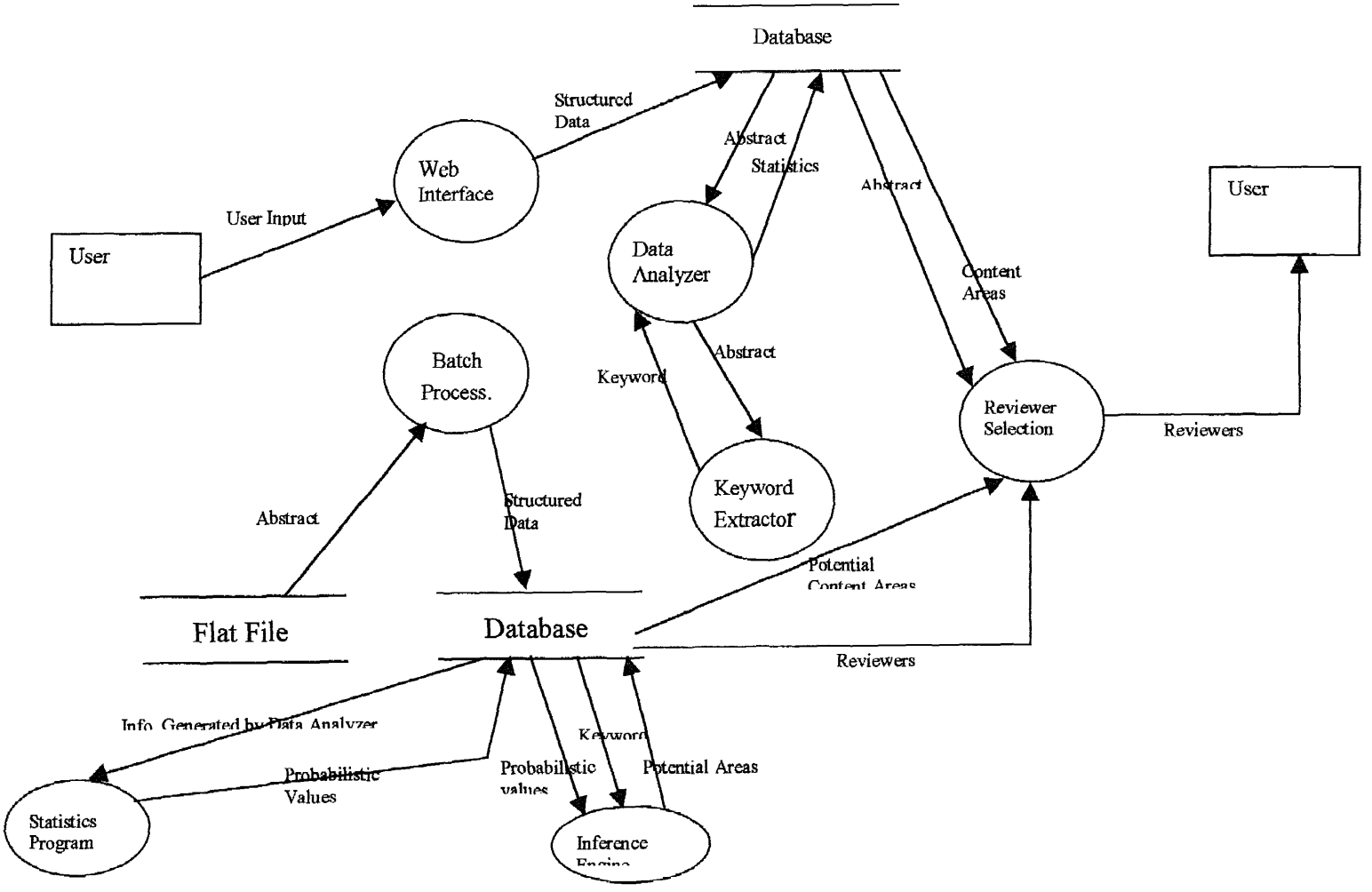

Figure 1 Functional Model

implementation of the network, was possible by the use of an API (Application Program Interface) called Hugin, which , supports all the different components of Bayesian Networks.

CHAPTER 2

BAYESIAN NETWORKS

Also known as belief networks, knowledge maps, probabilistic causal maps, It is a method of reasoning using probabilities. Judea Pearl gave its actual name in 1988, but the ideas are a product of many hands. Application of this theory has not only been accepted in Computer Science topics, but in others such as medicine (medical diagnosis), marketing and customer support, and robotics. Companies such as Microsoft and Hewlett Packard have applied these methodologies for developing outstanding intelligent software programs.

An expert system, is a system that analyzes the state of the world and based on its interpretation, decides on an action. Then the system expects some results from the action, which it will make come true or not, the results are used as a feedback to the same system. For instance, a physician examines a patient, and gets a record of symptoms, then she/he will conclude in a diagnosis and prescription of medicaments to cure the disease. After the prescription is applied the physician gets feedback from the patient visits and then she/he reformulates his/her prescription.

The first computer-based expert system was constructed in 1960. These systems were based in computer models of an expert such as : doctor, engineer, and mechanic. The constructs for these systems were production rules. A production rule is of the form.

If condition **then** fact or **if** condition **then** action.

These systems, then were called rule based systems, consisting of a knowledge base[1] and an inference system[2]. The knowledge base is a set of production rules and the inference system combines rules and observations to come up with conclusion about the state of the world. The only drawback of these models was when conditions were not totally assured, then results were not 100% accurate. Some modifications were proposed about this matter and one of then was to use production rules adding a certain probability value of belief. These rules are of the form :

If condition with certainty x **then** fact with certainty f(x).

A Causal or Bayesian Network, is a DAG (Direct Acyclic Graph) composed of nodes that represent random variables, and arcs that represent the relation of causality between nodes, these arcs are also called the independent assumptions of the network. When there is an arc or link from Node A to Node B, it is said that A is the parent of B, and in the same way B is a child of A. In Figure 2, there is an example of a small network representing the case when a person on his way home, wants to know whether his/her family is in. First of all, if the family is out, then the dog is taken to the yard and sometimes when the dog is at the yard, her bark can be heard. The family also turns on the light when they leave, but they also turn it on when they are awaiting a visit. The dog is also taken to the back yard when the dog has bowel problems. As you can see from the diagram, arcs represent relation of causality like in the case of "family out", then the "dog is out". This type of reasoning can be expressed in a more formal way as follows :

The Event A causes with certainty x the event B. (causation statement)

---

[1] Knowledge base : facts, assumptions, beliefs, heuristics, and expertise, in this project we refer to the information stored in the database.

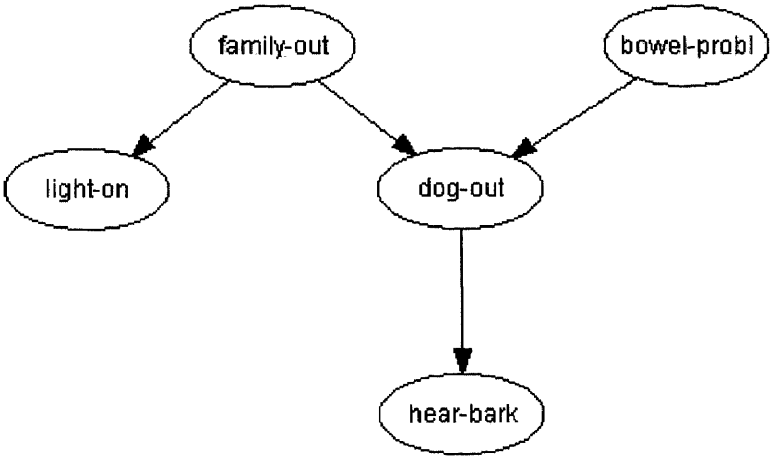

Figure 2 A Causal Diagram

In the same way, given the fact that the dog is out, one would like to know if this is because the "family is out" or the dog has "bowel problems". In this case the two root nodes are dependent to each other. This is called conditional dependence and the fact can be expressed in a more formal way as follows :

If we know that A has taken place, then B has taken place with certainty x.(reasoning statement).

As described before, each node is represented by a variable, which represents events (propositions). In some cases these variables represent Boolean events, such as in the previous example where every node represents a Boolean event (i.e. The dog is out of the house or not). Each event value is called a state, so in the case of Boolean variables, there are two states, but variables can have more than one state. A variable, may, for example, be the color of a car (states blue, green, red, brown) or a disease (states : bronchitis,

---

[2] Inference system: the process by which lines of reasoning are formed.

tuberculosis, lung cancer). At any time a variable is in one state, which it may be unknown. When a state is known in a certain point of time is called evidence.

2.1. Type of Connections

2.1.1. Serial Connections

It is a cascaded influence between Nodes. From Figure 2, Bowel problems influences dog out, which at the same time, one can hear the dog barking. In a formal way this is expressed as given three nodes : A,B, and C. if A influences on B, which in turn influences on C, is a serial connection. Obviously, evidence on A will influence the certainty of B which then influences the certainty of C. If the state of the intermediate node is known , then the channel is blocked, and A and C, in this case become independent. A and C are said to be d-separated and B is called an instantiated variable.

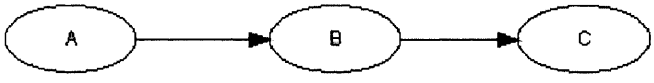

Figure 3 Serial Connection

2.1.2. Diverging Connection

Influence can pass between all the children of A unless the state of A is known. Nodes are d-separated only and only if A is instantiated.

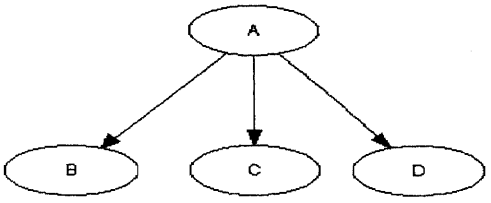

Figure 4 Diverging Connection

2.1.3. Converging Connection

In this case, If nothing is known about A, then the parents become independent. In the example of figure 2, if the fact that the dog is out is uncertain, then nothing can be assumed about the state of whether the family is out or the dog has bowel problems.

If any other kind of evidence, influences the certainty of A, then the parents become dependent . The evidence may be direct evidence on A, or it may be evidence from a child. This phenomenon is called conditional dependence.

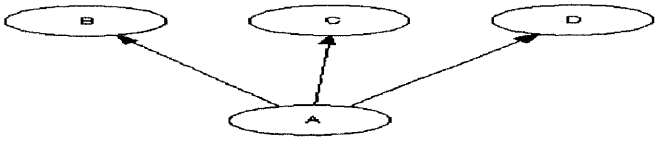

Figure 5 Converging Connection

2.2. D-Separation

Two variables A and B in a causal network are d-separated if for all paths between A and B there is an intermediate variable V such that either : the connection is serial or diverging and the state of V is known or the connection is converging and neither V nor any of V's descendants have received evidence.

If A and B are not d-separated, they are called d-connected. The implication of these definition denote that if A and B are d-separated, changes on the certainty of A have no impact over the certainty of B, and vice versa.

Before proceeding with the quantitative formulation of Bayesian networks, it is required to formulate the basic principles of this theory. This principles are based on classical probability calculus.

2.3. Basic Axioms

The probability P(A) of an event A is a number in the unit interval [0,1]. Probabilities obey the following basic axioms.

i. P(A) = 1 if and only if A is certain.

ii. If A and B are mutually exclusive, then

$$P(A \vee B) = P(A) + P(B).$$

2.4. Conditional Probabilities

A conditional probability statement is of the following kind :

Given the event B, the probability of the event A is x.

The notation for this statement is : P(A|B) = x. This notation is strictly read as *If B is true, and everything else known is irrelevant for A, then P(A) = x.*

The fundamental rule for probability calculus is the following:

$$P(A|B)P(B) = P(A,B) \quad (1)$$

Where P(A,B) is the joint probability of the event A and B. Conditioning equation 1 to a context C, then we get the following formula :

$$P(A|B,C)P(B|C) = P(A,B|C) \quad (2)$$

By analogy, from equation 1 we have the fact $P(B|A)P(A) = P(A|B)P(B) = P(A,B)$, which yields to the well known Bayes' rule:

$$P(B|A)=\frac{P(A|B)P(B)}{P(A)} \quad (3)$$

Bayes' rule conditioned by a context C will read,

$$P(B|A,C)=\frac{P(A|B,C)P(B|C)}{P(A|C)} \quad (4)$$

2.5. Likelihood

Likelihood of B given A is P(A|B), and it is denoted by L(A|B). It measures how likely it is that B is the cause of an event A. So, imagine different scenarios for different values of $B = B_1B_2...B_n$, then $P(A|B_I)$ calculates such a likelihood. If all B_is have the same prior probability, Bayes' rule yields

$$P(Bi|A)=\frac{P(A|Bi)P(Bi)}{P(A)}=kP(A|Bi)\,,$$

where k is independent of i.

2.6. Probability Calculus for Variables

As mentioned before, a causal network is composed of links and nodes, where each node represents a random variable which in a given time it is just in one state. This last sentence means the all states of a random variable are mutually exclusive and for each state, there is a probability associated to it.

$P(A)=(X_1,...X_n)$ where $X_I>=0$ and $\sum X_I=1$ for all i=1..n ,

X_I, denotes the probability of A being in state a_I.

Therefore, the probability of A being in state a_I is denoted $P(A = a_I)$ and denoted $P(a_I)$ if the variable is obvious from the context.

If there is a variable B, which states are : $b_1 \ldots b_m$, then the conditional probability P(A|B) is an n x m table that consists of a probability for each configuration (a_I, b_j), See table below.

Table 1 Conditional probabilities

	B_1	**B_2**	**B_3**
A_1	0.4	0.3	0.6
A_2	0.6	0.7	0.4

One can notice that the sum for each column is equal to one. From the conditional probability equation on equation 1,

$$P(a_I|b_j)P(b_j) = P(a_I,b_j) = P(a_I,b_j)$$

If we use the same table above to calculate P(A,B) with a vector $B_j = (0.3, 0.5, 0.2)$ and multiply each column, the result would be $P(a_I, b_j)$. See table below

Table 2 Joint Probabilities P(A,B)

	B_1	**B_2**	**B_3**
A_1	0.12	0.15	0.12
A_2	0.18	0.35	0.08

The sum of all cells gives the value of one. From a table P(A,B) 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 $(a_I, b_1),\ldots,(a_I,b_m)$. Then the conclusion is .

$P(a_I) = \sum P(a_I, b_j)$ for j = 1..m.

This calculation is called marginalization, and it says that variable B is marginalized out of P(A,B) (resulting in P(A)). The notation is

$$P(A) = \sum P(A,B) \tag{5}$$

From the previous example, we get the following values for P(A) = (0.39, 0.61).

2.7. Conditional Independence

Two variables are considered d-separated or independent when the following condition holds :

The variables A and C are independent given the variable B if

$$P(a_I|b_j) = P(a_I|b_j,c_k) \text{ for all I,j,k.} \tag{6}$$

This definition may look asymmetric ; however if equation 6 holds, then by the conditioned rule (Equation 4), we get :

$$P(C|B,A)=\frac{P(A|C,B)P(C|B)}{P(A|B)}=\frac{P(A|B)P(C|B)}{P(A|B)}=P(C|B)$$

As a summary of all the information presented, a Bayesian network consists of the following :

- A set of variables and a set of directed edges between variables
- Each variable has a finite set of mutually exclusive states.

- The variables together with the directed edges form a directed acyclic graph (DAG)[3].
- To each variable A with parents $B_1,\ldots,B_n$ there is attached a conditional probability table $P(A|B_1,\ldots,B_n)$.

If a node A has no parent, then this node does not have conditional probabilities, but instead, it has initial values, which are called unconditional probabilities.

2.8. The Chain Rule

Let P(U) be the joint probabilities of all variables in a Bayesian network. $P(U) = P(A_1,\ldots,A_n)$ where U is the universe of variables: $U = (A_1,\ldots,A_n)$. From this joint probability table, it is possible to calculate the individual probabilities $P(A_I)$ as well as $P(A_I|e)$, where e corresponds to an evidence. However the computation of $P(A_I)$ grows exponential in the way the number of variables increase, and therefore computation takes longer. There should be a more compact way of calculating P(U) without the exponential time factor and it is by the use of the conditional probabilities from the Bayesian network.

Theorem: (The chain rule) Let BN be a Bayesian network over

$$U = (A_1,\ldots,A_m),$$

Then the joint probability distribution P(U) is the product of all conditional probabilities specified in BN:

$$P(U) = \prod_i P(A_i \mid pa(A_i))$$

Where $pa(A_i)$ is the parent set of A_i.

---

[3] A DAG is characterized because there is no directed path $A_1 \rightarrow \ldots \rightarrow An$ such that $A_1 = A_n$

2.9. Probability Updating in Joint Probability Tables

Let A be a variable with n states, A finding on A is an n-dimensional table of zeros and ones.

Semantically, a finding is a statement that certain states of A are impossible.

Now, let U be a universe of variables, and assume that we have easy access to P(U), the universal joint probability table. Then, P(B) for any variable B in U is easy to calculate:

$$P(B) = \sum_{U\setminus[B]} P(U).$$

Suppose we wish to enter the above finding. Then P(U, e) is the table resulting from P(U) by giving all entries with A not in state i or j the value zero and leaving the other entries unchanged. Again, P(e) is the sum of all entries in P(U,e) and

$$P(U \mid e) = \frac{P(U,e)}{P(e)} = \frac{P(U,e)}{\Sigma_U P(U,e)},$$

Note that P(U,e) is the product of P(U) and the finding e. If e consists of several findings $(f_1,\ldots,f_m)$ each finding can be entered separately, and P(U,e) is the product of P(U) and the findings f_i. This can be expressed in a more formal way as follows :

Theorem : Let U be a universe of variables and let e = { f1, ..., fm}. Then

$$P(U,e) = P(U).f_1.\ldots\ldots f_m \text{ and } P(U|e) = \frac{P(U,e)}{P(e)},$$

Where

$$P(e) = \sum_U P(U,e).$$

Following, there is an example of a Bayesian network and how calculation are done. The Bayesian network in Figure 2, would be used and the problem statement is the following:

Let us suppose that Dr. Mason wants to know if by the time he comes back home at night, his family is at home before trying the doors. He has some clues that may help him guess in advance : Often when his wife leaves the house, she turns on an outdoor light. However, she sometimes turns on this light if she is expecting a guest. Also, there is a dog, that when nobody is home, it is put in the backyard. The same is true if the dog has bowel troubles. Finally, if the dog is in the backyard, he will probably hear her barking, but sometimes, he can get confused by other dogs barking.

After the statement is present, the next step is to model the network. As mentioned before this example is represented by the model in Figure 2. From the model we find out that we have two unconditional probabilities : family-out and bowel-problems, and the rest of nodes require conditional probabilities such as : P(light-on | family-out), P(dog-out| family-out, bowel-problems), P(hear-bark | dog-out). For each node we have two states with the values or true-false. So in the case of family out is true, we give a value of 0.30, and in false case we give a probability value of 0.7. This means that 70% of the cases, the family is at home. For bowel problems, a probability of 0.20 and 0.8 is predefined for each case. For the conditional probabilities, as a matter of example, let us take the case of P(light-on | family-out).

$$P(\text{light-on} = \text{true} \mid \text{family-out} = \text{true}) = .79$$

This means the probability that the light is on, given that the family is out. In these case, the value is 0.79. The rest of values and the tables for the rest of conditional probabilities are shown below :

Table 3 Conditional probability for Light-on

	Family-out = false	**Family-out = true**
Light-on = false	0.92	0.12
Light-on = true	0.08	0.88

Table 4 Conditional probability P(dog-out\|bowel-problem,family-out)

Bowel-problem	**False**		**True**	
Family-out	**False**	**True**	**False**	**True**
False	0.95	0.02	0.06	0.005
True	0.05	0.98	0.94	0.995

Table 5 Conditional probability for Hear-bark

	Dog-out = false	**Dog-out = true**
Hear-bark = false	0.6	0.46
Hear-bark = true	0.4	0.54

From this table, one can deduce that the number of probabilities to compute for each node is proportional to the number of nodes and the number of states of each node. So the formula can be expressed as : # of states for node i = ∏#of states of Node j, for all parents nodes of node i, including it itself.

Suppose that when Mr. Mason is very close home, he finds out that the outdoor light is on, so we entered this value as evidence and propagate the results. The results are shown in the following graphic

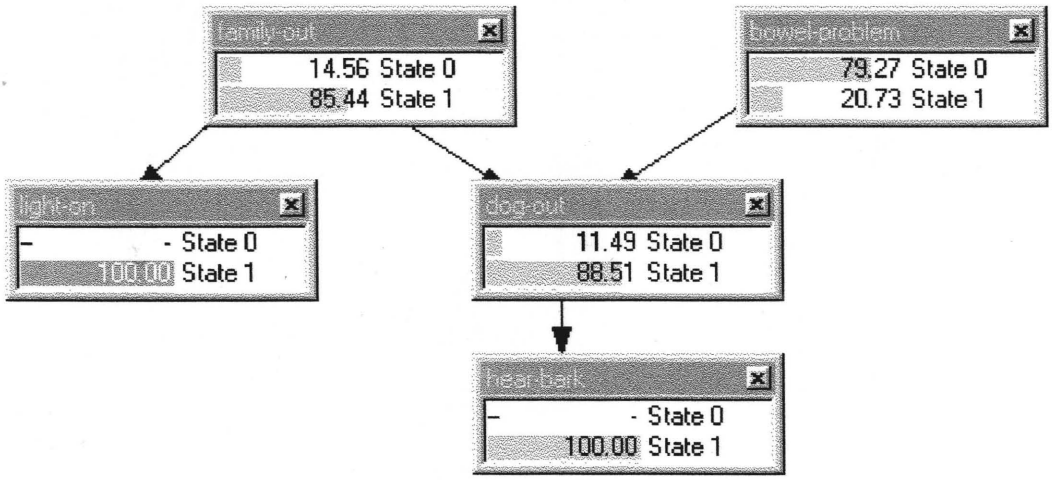

Figure 6 Propagation of evidence (outdoor light is on)

From the original unconditional probability values for family-out (0.7, 0.3), they have now become 0.1456 for the fact that the family is in, and 0.8544 for the fact that the family is out. What this shows that the evidence that the light is on, increments the probability that Mr. Mason's family went out.

Now, since Mr. Mason suspects that his family is away, he walks trough the alley to reach the backyard to confirm his suspicions. Finally he found the dog outside, and therefore he is pretty sure that his family is not in. (See figure below).

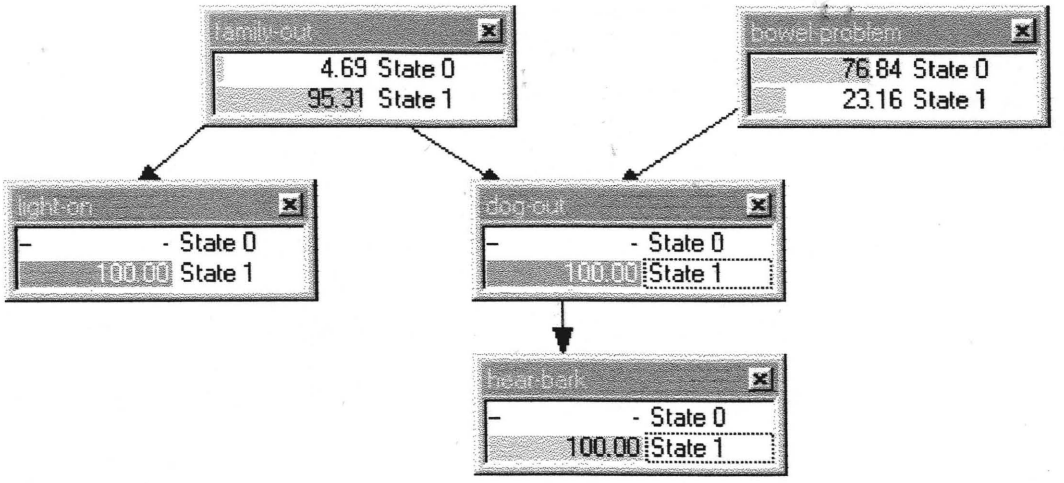

Figure 7 Propagation of evidence. (Dog is outside the house).

As the graphic suggests, the probability of family-out, has increased from 0.8544 to 0.9531 since the last evidence, which confirm the certainty that the family is out. Also from the bowel-problem node, the value of the probability of the dog, not having bowel problem is 0.7684, this is because the belief that the dog is out is because the family is out since the outdoor light is on.

CHAPTER 3

PROJECT DESCRIPTION

This project consists in the implementation of a Bayesian network that will be used to determine content areas of Artificial Intelligence documents. This network is composed of an ontology of all Artificial Intelligence areas and significant keywords that are indicative of a paper belonging to particular areas. Since a Bayesian network represents a causal relationship between its components, the causal relationship between the different Artificial Intelligence topics is represented upside down with respect to that of the Ontology; influence is explained as pointing from specific to generic subject. Additionally, there is a causal relationship between AI content areas and keywords. At the end of this chapter I will use a small example of the network to explain how the network is constructed and show the results after inferencing. The ontology is read from an ASCII file where the classification is represented in terms of indentation, AI is displayed at the first line and flushed completely to the left. The next levels in the classification are indented with respect to the previous ones.

As we know, Bayesian networks are initialized with probability values, so this information is extracted from a knowledge base composed of one Oracle table, called weight. The weight table is used to determine the degree of association between a keyword and an AI subject; a value called weight, is computed based on statistical formulas. According to a threshold weight value, keywords are imported into the Bayesian network, and these values are used to generate the conditional probabilities required for the network. After keywords have been incorporated into the Bayesian

network, then the network can be used for inferencing. The process starts by inputting the abstract or paper from a front-end interface such as a web page or a batch file, relevant keywords are extracted from the document and then passed into the Bayesian network to be entered as evidence. Keyword nodes that match those found in the document are set to one, and the keywords not present are set to zero. Then, this information is propagated throughout the network. Another program will check for content areas with high probability values. These high probability values mean a strong relationship between the content area and the evidence just entered. The content areas selected are then displayed as the content areas of the abstract or document.

3.1. Knowledge Base

The idea of this program is to give valuable data to the inferencing system in order to have good predictions. Valuable data, in the case of REVIEWER-SELECTION means a method or formula to determine the association that exists between a keyword and a particular content area.. Traditional probabilistic methods such as frequency of occurrence can not work in this case, since they would only calculate how many times a keyword will occur in a document or in other words, the frequency of occurrence of words inside AI documents. After certain period of investigation We based our approach on that used by Salton[4] for a similar problem : the problem of indexing terms to document content for easy retrieval, in how to identify what is relevant and what it is not relevant. He, then figured out, that in written text, grammatical function words such as "and", "of", "or", and "but" exhibited approximately an equal frequency of occurrence in

different documents of different contents. So he also analyzed the case of nonfunction words, and he found out that there were extreme variations in frequency and that in some way those variations were related to the document content. He concluded then, that the frequency of occurrence of non-function words may be used to indicate term importance for content representation. Then he proposed the following algorithm to extract valuable term from documents :

1. Eliminate common function words from the document texts.
2. Compute the frequency for all remaining terms
3. Choose a threshold value T , to select those terms with a value greater or equal than T and reject the rest.

There was a problem with this algorithm, using frequency of occurrence, was not a very selective way to determine association between terms and content areas. This is true because when calculating frequency for a keyword that occurs in more than one content area, there would not be any difference to that one of a keyword that occurs in only one document. So there should be a way to punish those terms or keyword that occurred in more than one content and in some way to increase the value of those terms that appeared in documents of a specific content. Salton, found similarities in the concept weight and his observations, so he developed the following formula :

Let Wij, be called weight or importance of a term or keyword Tj in a document Di, and is defined as the keyword frequency multiplied by the inverse document frequency.

$$W_{ij} = tf_{ij} \cdot \log \frac{N}{df_i} \qquad (7)$$

---

[4] The name of the book is : Automatic Text Processing , the author is : Gerard Salton.

For the purpose of this thesis, we have adapted this formula in the following way :

$\mathbf{W_{ij}}$ = will determine the weight or in other words, the closeness of association between a particular keyword i and a particular area j.

$\mathbf{IR_{ij}}$ = Indicator ratio. = $\frac{\text{\# of papers of area j in which keyword i occurs}}{\text{Total number of papers of area j}}$

$$W_{ij} = IR_{ij} * \log\left(\frac{\text{\# areas j}}{\sum IR_{IJ} \text{ for all j}}\right) \quad (8)$$

The values for IR_{ij}, will be supplied from the data analyzer which outputs to an Oracle database table: the id of the document, the keyword occurred in the document, the content area the document belongs to, and the number of times the keyword occurs within the document.

3.1.1. Functional Specification

Following is the flowchart diagram of the statistical program.

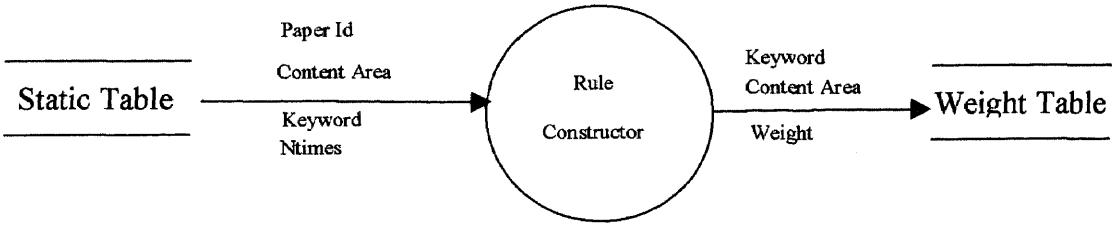

Figure 8 Flow Chart Diagram – Rule Constructor

The program receives as input : PaperId which is the identifier that uniquely distinguish each document, Content Area, the area the document belongs to, Keyword , is the keyword found in the document and extracted as a relevant keyword from the keyword extractor, Ntimes, number of times the keyword happens in the document. Then

the Rule Constructor generates the weight for each combination of keyword-are and outputs the result into the Weight table.

3.1.2. Process Description

This module generates a weight value which represents the degree of association between keywords are Artificial Intelligence content areas. The weight value is computed by using Equation 8.

Begin
Query = SELECT DISTINCT AREA,KEYWORD FROM STATIC
For each record in Query do
  Extract area and keyword in areal, keyword1
  Compute IR_{ij} as number of paper where area = areal and keyword = keyword1 divided by total number of papers of area = areal
  Compute second term of the formula by summing all IR_{ij} that have as a keyword = keyword1. Assign this value into sumAllIRij.
  $Weight_{ij}$ = IR_{ij} * log(Total number of areas/ sumAllIRij)
  Store $Weight_{ij}$, keyword1, and areal in Weight table
  Read next record from Query
End for
End process.

Figure 9 Algorithm for the Rule Constructor program

3.1.3. Design

Program was implemented using Java 1.2 and information was extracted from Oracle tables corresponding to the database model for this project. In Java, a special module called JDBC was used, this module allowed the connectivity between Java and Oracle. This program is not part of the interactive module, it works in batch mode and it is run every time the information in the static table is changed. The source code of this program can be found in Appendix B of this document.

3.2. Inference Engine System

The inference engine system has to deal with the creation of a Bayesian network in order to automatically derive content areas from keywords contained in a paper. The structure of the network is composed of content areas supplied from Artificial Intelligence and keywords extracted from the rule constructor. Artificial intelligent subjects are classified in a way of a DAG (Directed Acyclic Graph). In this type of network, specific nodes are not associated to one general subject; a specific subject may be related to more than one subject area. So at the highest level of the hierarchy, there are the most specific classified subject areas and at the lowest level, there is the AI subject with all the immediate higher-level subject areas connected to it. Information about the content areas of AI, is stored in a text-file, in which, for each line, there is a subject category and there is an indentation that represents the relationship between the subject in the previous one and the current one. With regards to keywords, not all keywords in the weight table are imported in the Bayesian network, only those keywords with a weight greater than a threshold value will be used. As a summary of the representation of the Bayesian network, areas will be in the top level configured in a way of a DAG and at the bottom level , keywords are located. The way keywords are connected to areas depends on the weight values, so there will be some cases where a keyword is connected to just one area , or other cases where a keyword is connected to more than one. A tentative diagram is shown in Figure 10.

Once areas are extracted from the text file and keywords are imported based on the threshold value from the static oracle table, the next point would be to assign probabilities to these nodes. All these nodes are Boolean nodes, what they express in the

case of an area, is the probability of certainty of a content area based in a more specific area, and in the case of a keyword node is the probability of certainty of a keyword happening in a document of that type. The conditional probabilities, such as in the case of the keyword "superset" in Figure 10, may express "Probability of the keyword 'superset' happening, given that this paper is classified in 'coalitions' but not in 'AI architectures' ". Those nodes with no parents are initialized inversely proportional to the number of orphans. In the case of areas, according to the structure, an area is usually influenced by more than one node (multiparent). Therefore probability tables are usually big, and for reasons of simplicity I assumed a probability value of 1 if any of the parents (influencing nodes) occurs in the document. When computing the case when none of the parents occur in the document , I assume a probability of 0 and 1 for the case of "yes" and "no" respectively.

Keywords have a different way of assigning probabilities with respect to areas since these can have more than one parent. The only information provided for computing probabilities is the weight value, which is not enough, since in the case of one parent, it is required four different values (each value for each combination parent-node). The assumption taken from here is : add all the weights of all parents of the keyword node, for every combination probability the value would be the multiplication of the weight values of the nodes involved in the combination over the total sum of weight to the power of the number of parents. For instance, let us use the keyword "superset" from Figure 10 and assume we have the following weight values :

Table 6 Weight values for the "superset" keyword node.

Area	Weight
AI architectures	4.50
Coalitions	3.08

Total sum = 4.50 + 3.08 = 7.58

Since we have three nodes involved and each node has two states, we have $2^3 = 8$ different probability values to assign to the keyword node "superset". Here are the probability values for this particular case :

Table 7 Probability values for the keyword node "superset"

AI-architecture	**False | No**		**True | Yes**	
Coalitions	**False|No**	**True|Yes**	**False|No**	**True|Yes**
False|No	1	1 – 0.40=0.60	1-0.6=0.40	1 – 0.24 = 0.76
True|Yes	0	3.08/7.58 =0.40	4.5/7.58=0.6	$(3.08*4.5)/7.58^2 = 0.24$

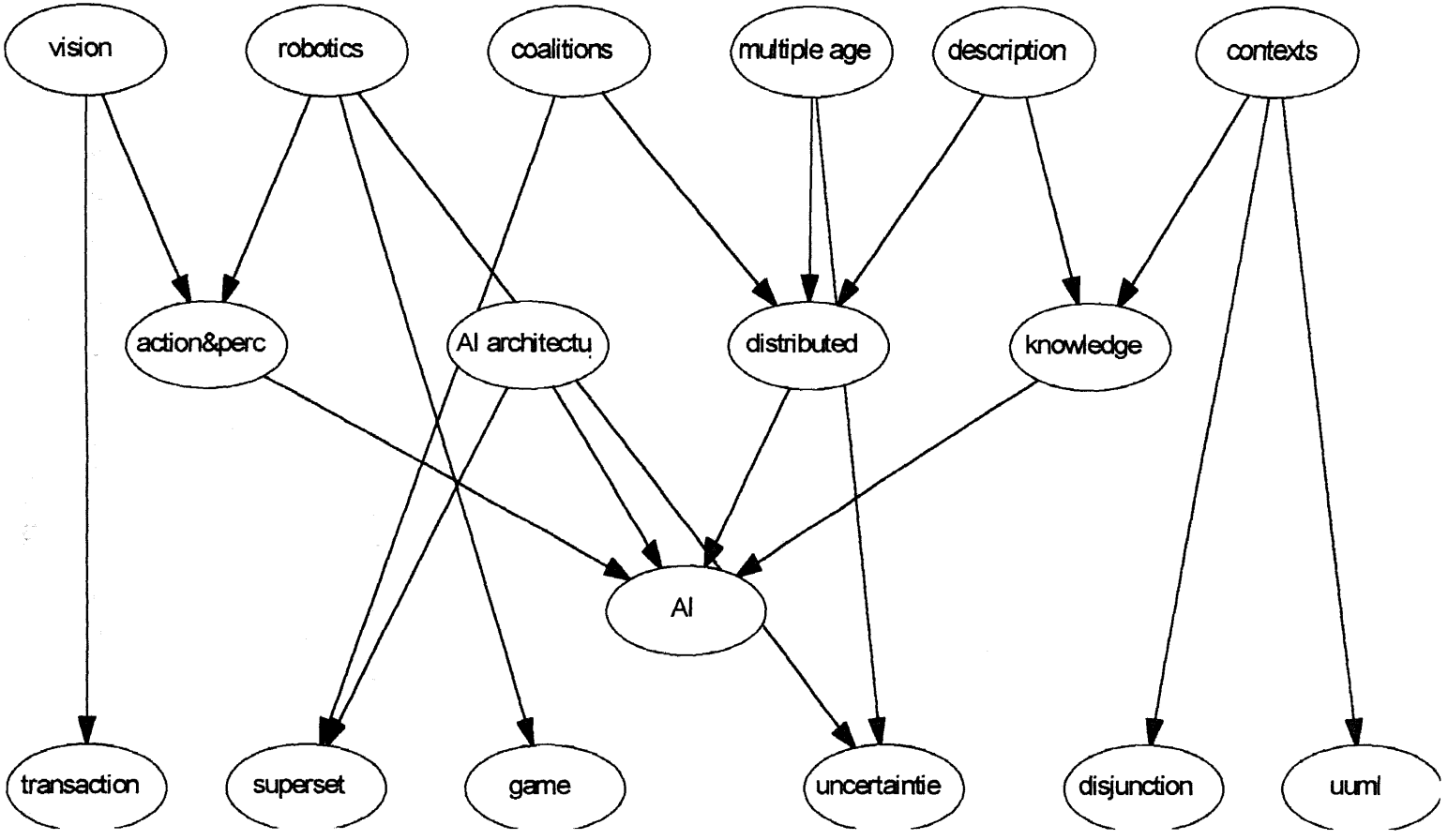

Figure 10 A causal diagram for Reviewer-Selection.

In the scenario where AI-architecture = No, Coalition = Yes and Superset = Yes, it is noticed that the only node affecting is Coalition, so then, the probability value is the division of its weight over the total sum of all weights. From the first column, also notice that probability of keyword "superset" happening, given that the document is not classified as AI-architecture, neither Coalition is 0. This value is because it is assumed that a keyword can not appear in a document of any other subject than the one it is related to in the Bayesian network.

After all probability values have been assigned, the network can be used for inference. A new abstract is inputted into the web interface, and then this interface calls the keyword extractor to extract relevant words from the document. These keywords are then entered into the Bayesian network as evidence. Those keywords that are not present in the Bayesian network are taken out of the evaluation. After setting evidence on the different nodes, by using the Universal Joint probability, the evidence is propagated and the new probability values are drawn out. A program will scan probability values throughout the area nodes and those nodes with significant probability values are then displayed as potential subject areas for such document.

As a purpose of understanding, following, I will use a small example to demonstrate how the project works, starting from the Ontology ASCII file until the potential subject area that fits to the document is deduced after inference. Appendix C presents a small portion of the ASCII file as well as the different probability tables of every node in the network.

Figure 12 shows a small Categorization network, composed of the area nodes : AI, Distributed AI, Decision trees, real-time systems, search, planning, causality, description

logic, theorem proving and cognitive modeling, and keywords membership, real-time, pathways, simplex and epsilon. Probability values for subject areas are set to 1 for all the cases when any of its parents happens as explained before. Keyword are assigned probability values depending on the weight values extracted from the Weight table of those subject areas that influence it, therefore, in the case of epsilon, this node has two subject areas as parents : cognitive modeling and theorem proving. The weight values extracted from this table are 3.34 for cognitive modeling and 4.78 for theorem proving. Therefore as explained before, the probability of epsilon happening (being yes) given that the document is classified under cognitive modeling but not in theorem proving is equal :

Prob(epsilon = yes | cognitive modeling = yes & theorem proving = no) = 3.34 / (3.34 + 4.78) = 0.41

Apendix C contains all the input data to the Bayesian network : the ontology file, keywords extracted from the weight table and the probability tables of every node. After probability tables are ready with the values assigned for each combination case, the network is ready for inference. Let us suppose the system is tested with a document that contains the words "simplex", "real-time" and the rest of words are not present in the document. Once keywords are extracted from the document, they are entered as evidence in the system. In this case, then, the two keywords are entered as evidence, which means its certainty is set to 100 percent in the case of yes. For those keyword nodes whose keyword, do not occur in the documents, are set to zero percent in the case of True. Then propagation of these values is done throughout the network. The results of the propagation are shown in Figure 13. Analysis is started from the top level of the Bayesian network based on the fact that the more selection of specific subject will lead to more

accurate results. A threshold probabilistic value is assumed for purposes of selecting the potential subject areas that the document may fit. Therefore a value of 55 is chosen for this examples, so subjects nodes under that value are discarded, otherwise they are inserted in the list of potential nodes. By looking at the "yes" field values of the top level, the highest value is achieved by real-time systems with a value of 51 below the threshold. In this case since none of the nodes reach or overpasses the threshold, then selection is moved to the next level of specific subject areas. In the second level, the highest value is obtained by decision trees which is 56; this value is over the based value, therefore this subject is picked as a potential classification subject for the given document. Results in lower levels are discarded since they represent redundant information like in the case of decision trees and distributed AI where the later is a generalization of the earlier(i.e. the fact that the document fits in distributed AI is redundant, given that the document is already fitted under a subcategory of distributed AI). As a result of the inference, Decision Trees is displayed as the subject where the example document can be classified.

3.2.1. Functional Specification

The flowchart that represents the different programs that make up the Inference system is given in page 38. In the storage symbol, in the middle of the graphic, you can see the name "Hugin Knowledge base file", this is the file type used by Hugin to store networks.

The text file that contains the list of subject areas, represents the classification by means of indentation , so the more indented the file, the more specific the classification is. The "Graphical representation" process was not included in the functional

specification because of lack of relevance, this routine is more important for the presentation of results.

3.2.2. Process Description

Import Areas

This routine imports AI subject area names from a text file. The classification of these subject areas is represented in the text file as a matter of indentation. If a subject appears more indented that the previous one, it means that the subject is classified under this previous. The subject areas are then incorporated into the Bayesian network. A detailed specification of the algorithm is displayed in Figure 11.

```
Main subroutine
Begin
Create new domain d.
Open the text file with all the content areas.
Read the first line of text file
While not end of the text file Do.
  Extract area name from text line.
  Create new node in the B.N with the area name just extracted.
  Call recursive subroutine Import Subjects.
End while.
Save domain d
End subroutine.

Import Subjects subroutine.
Begin
Read next line from text file.
If it is not the end of file then
   If line is indented with regards to previous line then
      Extract subject name from text line.
      Create new node in BN with name just extracted from text file.
      Link this new node as child of node read from previous line.
      Make recursive call to Import Subjects subroutine.
    End if
End if
End subroutine.
```

Figure 11 Import Subjects subroutine

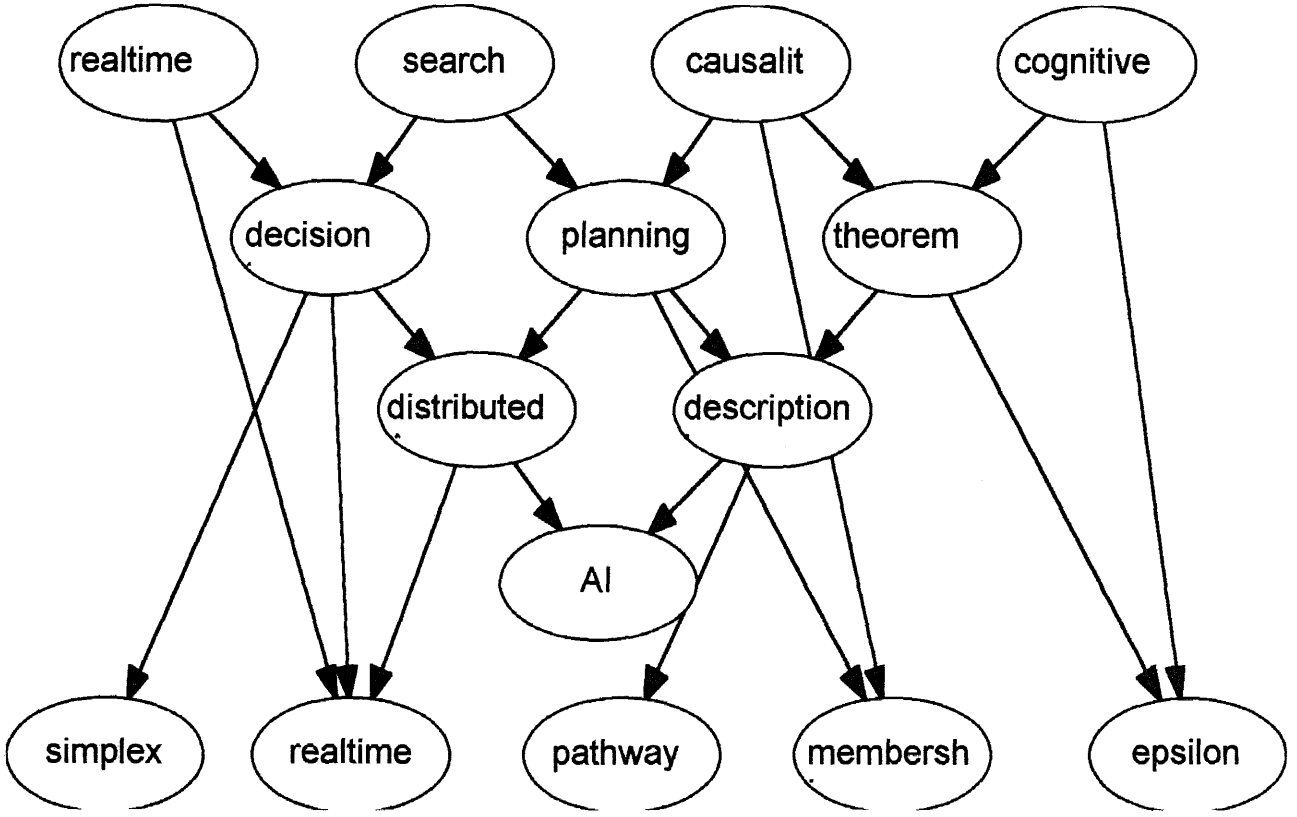

Figure 12 An example of a Categorization network

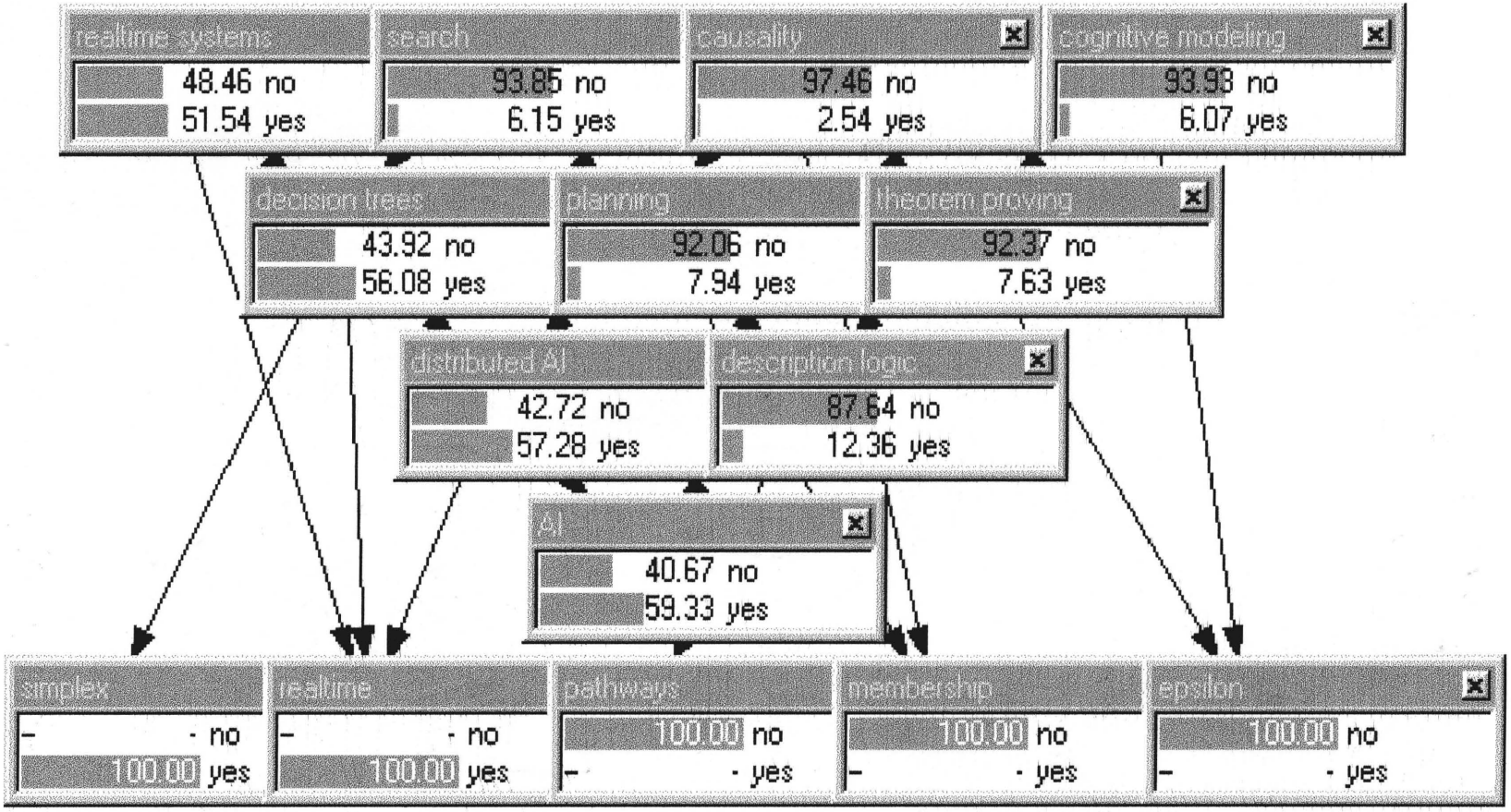

Figure 13 Propagation of evidence – Small example.

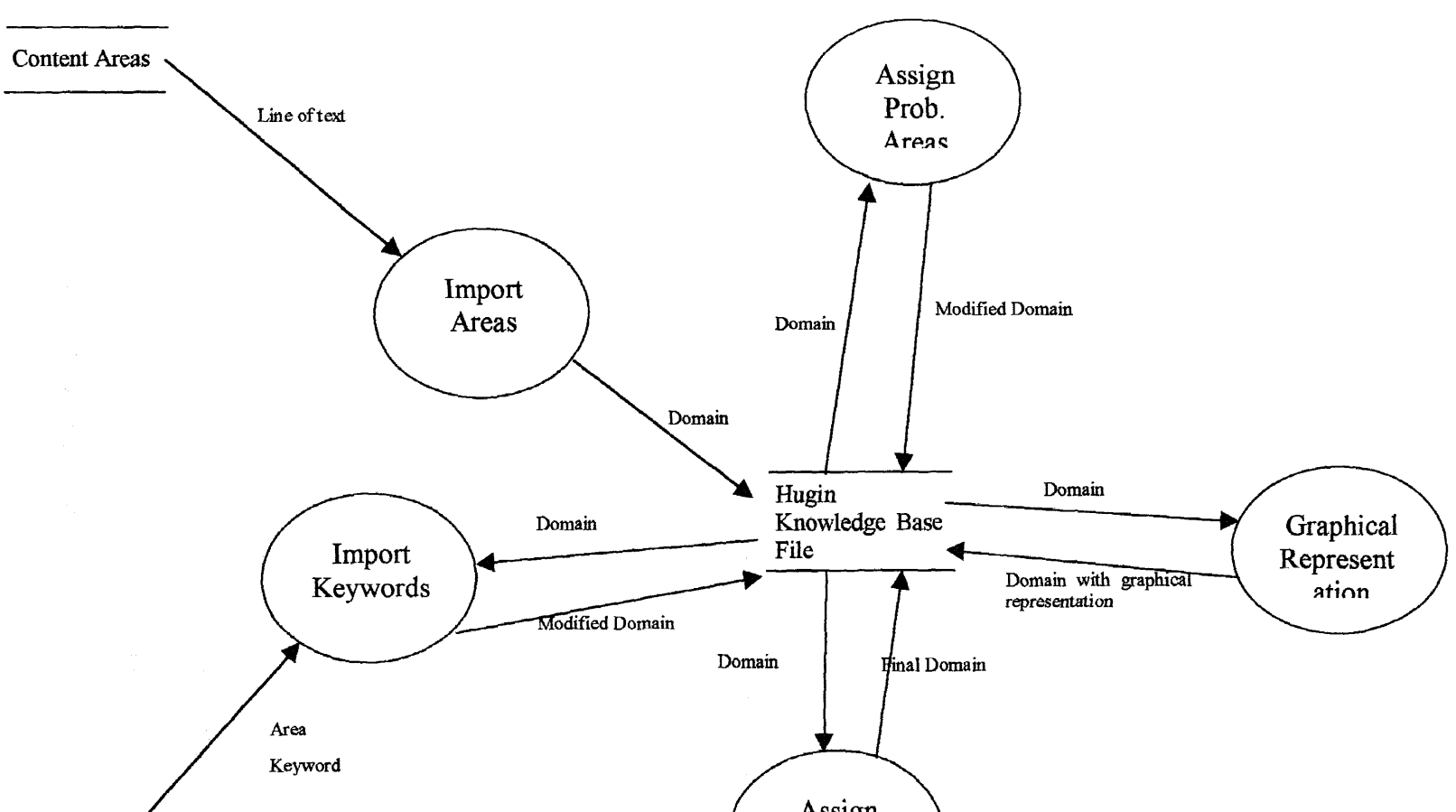

Figure 14 Flow chart Diagram Inference Engine System

Assign Probability to Areas

This subroutine assigns probability values to the imported areas into the Hugin file. The number of probabilities to compute, depends in whether a node in the Bayesian network has links or not. In the case of a root node, default values must be assigned, like in the case of the AI node, the probability values were 0.5 for each case. The other case presented in this network is a node connected to one parent in which the values are computed based in the number of children of the parent node. Figure Figure 15 displays the corresponding algorithm.

```
Main subroutine
Begin
Open domain d
For each node in the domain d Do
  If node is a root node then
     Probability values are 0.5 for both cases
 Else
    Find number of siblings for this node
    For probability value equal "yes", assign the value 1/ (number of siblings).
    In the case of "no", assign the value of 1 – 1/(number of siblings)
 End if
End for.
Save domain d with new changes.
End subroutine.
```

Figure 15 Main Subroutine : Assign probabilities to areas

Import Keywords

Keywords are extracted from the Reviewer Selection database, based on a threshold value. The information stored in the database is : keyword, area, and weight. The threshold value is compared against the weight to extract those pairs keyword-area with a weight bigger than threshold. Once keyword-area are drawn out, the keyword is inserted as a node into the Bayesian network and is linked to the area node corresponding to area

extracted from the database. The weight value is saved on the keyword node with the purpose of computing probabilities for the keyword later on. The algorithm is explained in the next figure.

```
Main subroutine
Begin
Get threshold value from command line
Get username and password from command line to connect to oracle database
Open domain d
Select_1 = "SELECT KEYWORD, AREA, WEIGHT FROM WEIGHT WHERE WEIGHT >
:threshold".
Connect to oracle database with username and password
Execute select_1
For each record extracted from select_1 Do
  Draw out keyword, area, and weight from record
  Look for area inside domain d
  If node found with same area name then
    Look for keyword inside domain d
     If node found with same keyword name then
        Link area node as parent of keyword node
     Else
         Create new node and give it keyword name in d.
         Link area node as parent of this new keyword node
     End if
      Insert weight and parent name as attribute inside keyword node
  End if
End for.
Save domain with new changes.
Close connection to oracle database.
End subroutine.
```

Figure 16 Main subroutine : Import keywords

Assign Probability to Keywords

After keywords are imported into Hugin, probability values need to be assigned in order to process the network. As different to the case of AI content areas, keywords can have more than one parent, which implies more probabilities to compute. Depending on the number of parents the number of probability values is equal to 2^{n+1}, where n is the

number of parents. Probabilities are then computed based on the AI content areas involved in the condition. After assigning probabilities, the Bayesian network is ready for inference. The algorithm is explained in next figure.

```
Main subroutine
Begin
Open domain d
For each keyword node inside domain d Do
  Find the number of parents of keyword node
  If number of parents = 1 then
    Probability values will be 0.3 for the case "no", and 0.7 for "yes"
  Else
      From the keyword node, compute the sum of the weight of all parents connected to this
      node
      Number of probabilities to calculate = 2 to the power of number of parents
      For i = 1 to the number of probabilities to calculate Do
         Convert i into binary format.
         From the binary format, determine which parents are involved in such a combination.
         For each parent involved in combination multiply the corresponding weight value and
         store it into num.
         At the same time multiply total sum as many times as parents involved in the
         combination and store into den.
         Probability for "yes" = num / den
         Probability for "no" = 1 – num / den.
      End for
  End if
End for
Save domain d
End subroutine.
```

Figure 17 Main subroutine : Assign probabilities to keywords

3.2.3. Design

The inference engine system is basically written in C language, and developed for the Unix Operating system. This system was built with the help of an Artificial Intelligent software tool called Hugin, which is specially design for the manipulation of Bayesian Networks (A demo version can be downloaded from their website http://www.hugin.dk). This software comes with two applications, one is the runtime program which is used for

the graphical manipulation of networks, and the other is an API[5] library developed in C which provides the same functionality as the runtime program.

Following is a brief description of all the different functions from the API version 4.0, used in the development of this project.

In order to use the Hugin API, you need to include the library <hugin.h> in your C program and make sure that you have the environment variable $HUGINHOME set in your .cshrc file. To compile the program also be advised of the following command line :

%homer> cc –I$HUGINHOME/include –c myprogram.c

Hugin also handles the following datatypes :

- h_number_t : single precision floating point.
- h_double_t : double precision floating point.
- h_triangulation_method: defines the possible triangulation methods used during compilation.
- h_error_t : defines the various error codes returned when errors occur during execution of API functions.
- h_status_t : common return value of some API functions. If value is zero, the function succeeded; if the value is nonzero, the function failed and the value will be the error code.
- h_string_t : used as a character string data type.
- h_domaint_t : domain data type. A domain is the name given from Hugin to the whole network
- h_node_t : Node data type. Node of a BN network

---

[5] Application Program Interface

Here it is a brief description of most of the API functions used in our project :

h_domain_t h_new_domain(void) : creates a new empty domain. If creation fails, NULL is returned.

h_node_t h_domain_new_node(h_domain_t domain, h_node_category_t category, h_node_kind_t kind) : Create a new node of the indicated category and kind within domain.

h_status_t h_node_delete(h_node_t node) : Remove node (and all links involving node) from the domain to which node belongs.

h_status_t h_node_add_parent(h_node_t child, h_node_t parent) : Add node parent as a new parent of node child.

h_node_t * h_node_get_parents(h_node_t node) : Return a NULL-terminated list comprising the parent nodes of node.

h_table_t h_node_get_table(h_node_t) : Return the probability table associated with node. If the node is a chance node, the table is the conditional probability table for node given its parents.

h_node_t h_domain_get_first_node(h_domain_t domain) : Return the first node of the domain. The first node is based on the most recent created node. This is with the purpose of traversing.

h_node_t h_node_get_next(h_node_t node) : return the node that follows node according to the date of creation.

h_status_t h_node_set_attribute(h_node_t node, h_string_t key, h_string_t value): Insert a new attribute into node. This was used in the thesis with the purpose of storing the weight values of the parents of node.

h_string_t h_node_get_attribute(h_node_t node, h_string_t key): Return the value associated with key in the attribute list for the node.

h_status_t h_domain_save(h_domain_t domain, h_string_t filename, h_endian_t format) : Save the domain as a HUGIN kb to a file named filename.

h_domain_t h_load_domain(h_string_t filename) : Load a domain from the HUGIN KB file named filename.

h_node_t * h_table_get_nodes(h_table_t table) : Retrieve the NULL-terminated list of nodes assoicated with the table. If an error is detected, NULL is returned.

h_number_t *h_table_get_data(h_table_t table): Retrieve a pointer to the array of table holding the actual discrete data (denoted by x[i]). This array is a one-dimensional representation of the multi-dimensional array. It is possible to modify the contents of the table from this pointer.

size_t h_table_get_size(h_table_t table): Return the size of the table. If an error is detected, (size_t) –1 is returned.

h_status_t h_node_set_subtype(h_node_t node, h_node_subtype_t subtype): Set the subtype of the node to subtype. h_node_subtype_t can be any of the followings : h_subtype_boolean or h_subtype_label. By default is set to h_subtype_label.

h_status_t h_domain_write_net(h_domain_t domain, FILE * net_file) : produce a specification of the Hugin kb file in text format using a native language called NET.

In order to import the keyword to the Hugin Bayesian network was necessary to use Pro*C, a C language for Oracle since information was stored in an Oracle Table. The rest of the programs were written in C including the HUGIN library.

CHAPTER 4

CONCLUSIONS

The planned objectives from the beginning of the project were accomplished to the fullest. Even this particular case of categorization of documents for Artificial Intelligence, could be extended for the classification of documents in other disciplines by means of supplying the respective ontology of the discipline and the correct assignment of probability values to the association between keywords and the elements of the ontology.

The success of the thesis is reflected in the following attained achievements:

- The construction of the Bayesian network through the extraction of the Artificial Intelligence ontology from an ASCII file and the importation of keywords from an Oracle database based on a threshold value. There is no limitation from the network with regards to the number of components (whether it is a keyword or content area).
- The creation and implementation of a statistical formula that determines the degree of association between a keyword and a content area. This formula allows deriving the probability values required to feed the Bayesian network.
- Portable and network independent algorithms for the assignment of probability values. The routines utilized for the assignment of probability values can be used in any type of network configuration, also they interface with varied types of software languages such as Java, Pro*C and C language.
- The graphical representation of the Categorization network. This was achieved by the use of the Hugin API and can be displayed using the Hugin runtime program.

APENDIX A

USER MANUAL

The expert system is composed of six program files written in Java, Pro*C and C. There is also an oracle table called "weight" used as the knowledge base for the system. The Operating system over which the programs were created was Unix System V version 4.0. These programs are not user interactive, they work in batch mode and parameters are given from the operating system command line. The program files are located in the server : logic.njit.edu and the path is : /home/challeng/william. In order to have access to the files, you need to telnet logic; if you are on campus, just telnet to logic, outside the University campus, telnet to logic.njit.edu.

The program files that use the Hugin API, do not run under the logic account because license was only purchased for one server which is : homer.njit.edu. So compilation and execution of such programs must be made from any homer account.

Program file : RuleConstructor.java

Description : This programs builds the knowledge base for the inference system. It generates and inserts weight values in the weight table. Currently, the program works for the shared oracle account challeng.

Language used : Java 1.2

How to compile : from the command line type the following line :

Logic% javac RuleConstructor

After compilation, a file RuleConstructor.java is generated.

How to run : from the command line type the following line :

Logic% java RuleConstructor

Table description :

Name : Weight
Fields : AREA VARCHAR2(35), KEYWORD VARCHAR2(25),WEIGHT NUMBER(8,4)

Program file : import_a.c

Description : This program imports the AI subject areas from a text file.

Language used : C language.

How to compile : This program only compiles in Homer, since Hugin API is installed there. There is a file that contains all the compile instructions for this program : compimp. This file generates the executable called import_a. So in the command line type the following :

Homer% compimp

How to run : There are two arguments for this program, one is the text file with the AI subject areas, which for this case is called INPUTA.DAT. Notice that the file must be typed uppercase because Unix is case sensitive. The other argument is the output Hugin file where areas will be stored.

Homer%import_a INPUTA.DAT bayesnet

Program file : builprob.c

Description : Assign probabilities to subject areas. This program is dependent of the network structure. So it only works with the current structure.

Language used : C language.

How to compile : There is a compile file to execute, called compbuild. It creates the executable : buildp. This program as well as the previous one can only be compiled and executed in Homer.

This is the line to compile it :

Homer% compbuild

How to run : It requires only one argument which is the name of the Hugin file. The Hugin file should contain all the subject areas at this point. Here it is the command line :

Homer%buildp bayesnet

Program file : impkey.pc

Description : It extracts keywords from the weight oracle table to the Hugin file, according to a threshold value given as a parameter to the program. It also links keywords to AI subject areas based on the information stored in the table.

Language used : Pro*c precompiler

How to compile : Since this program uses Hugin API library functions, it can only be compiled in Homer. There is a makefile file with all the compilation instructions for this program. The file is called test.mk. From the command line type the following :

Homer%make –f test.mk EXE=impkey OBJS=impkey.o

After compiled, an executable file called "impkey" is created.

How to run : The parameters are :

1. Threshold value : starting value for importing keywords into Hugin file.
2. Oracle username : Oracle name
3. Oracle Password : Oracle password
4. Hugin file : Hugin file which should include the AI subject areas.

A typical command line should look like this :

Homer%impkey 4.50 challeng@logic chal101 bayesnet

Program file : assignkp.c

Description : Assign probability values to keywords after they have been imported from the weight table

Language used : C language

How to compile : There is a compile file called compass. This compilation only works in Homer.

Homer%compass

How to run : There is only one parameter which is the Hugin file. This file at this point should include the keywords imported from the oracle table. The command line is :

Homer% assignkp bayesnet

Program file : drawnet.c

Description : This programs generates the graphical coordinates in order for the categorization network to be displayed in the Hugin Runtime module.

Language used : C language

How to compile : There is a compile file called compdraw. As well as most of the programs, it only compiles and runs in Homer. The command line to compile is :

Homer% compdraw

How to run : Drawnet requires three parameters : the name of the Hugin file which should be complete at this point, the width of the node given in pixels, and the height given also in pixels.

The command line should look similar to this :

Homer%drawnet bayesnet 50 30

In order to see the graphic representation of the categorization network, you need to run Hugin runtime which is also installed in Homer. It is required to have an account in Homer to run the Hugin runtime. The procedure for running Hugin runtime is :

1. Logon into a Unix machine that has Xwindows or OpenWindows
2. Open a shell command Window inside Xwindows or OpenWindows
3. Type the following command from the local host : xhost +homer.njit.edu
4. Telnet to homer.njit.edu with username and password
5. After being logged on, make sure you have set the environment variable HUGINHOME or that you include the following path in your command line : /usr/local/Hugin/bin.
6. In the command line type xhugin if you have the HUGINHOME variable set, otherwise type the full path : /usr/local/Hugin/bin/xhugin
7. Open the categorization file from the directory you have it stored.

APENDIX B

SOURCE CODE

RuleConstructor.java

```
/*************************************************************************/
/* Program : RuleConstructor                                             */
/* Subject : This program generates the rules to determine the           */
/* relation                                                              */
/* ship between area - keyword. For every tuple area-keyword a weight*/
/* is computed using an statistical formula.                             */
/* Input: Table with AREA,KEYWORD,PID,TIMES                              */
/* Output : Table with AREA,KEYWORD,WEIGHT                               */
/* Date : 01-12-1999                                                     */
/* version : 1.0                                                         */
/*************************************************************************/

import java.sql.*;
import java.lang.*;

public class RuleConstructor
{
 public static int ConstructRule()
 {
  Connection con;
  Statement s_1,s_2,s_3;
  ResultSet rs_1,rs_2;
  PreparedStatement ps_1;
  String query1,query2,insert_1,delete_1;
  int numberareas = 0;


  //update query to clean up the table

  delete_1 = "DELETE FROM WEIGHT";

  //Main query to extract area and keywords

  query1 = "SELECT DISTINCT AREA,KEYWORD FROM STATIC";

  // Query to retrieve the number of areas
```

```
query2 = "SELECT COUNT(DISTINCT AREA) FROM STATIC";

// Insert statement to update the table t_weightkeywarea

insert_1 = "INSERT INTO WEIGHT(AREA,KEYWORD,WEIGHT) VALUES (?,?,?)";


try{
    Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
   }
 catch(Exception e) {
    e.printStackTrace();
    return 0;
   }
try {
     con =
DriverManager.getConnection("jdbc:oracle:thin:@logic.njit.edu:1521:logi
                c40","username","password");
    }
     catch(SQLException exOb) {
     System.out.println("Error getting connection
                                          "+exOb.getMessage());
     return 0;
     }
try {
      //Table weight is cleaned
      s_3 = con.createStatement();
      s_3.executeUpdate(delete_1);

      s_1 = con.createStatement();
      ps_1 = con.prepareStatement(insert_1);
      rs_1 = s_1.executeQuery(query1);
      rs_1.next();
      s_2 = con.createStatement();
      rs_2 = s_2.executeQuery(query2);
      rs_2.next();
      numberareas = rs_2.getInt(1);
      while(rs_1.next())
       {
        float IRij;
        String sarea,skeyword;

        sarea = rs_1.getString(1);
        skeyword = rs_1.getString(2);
        IRij = ComputeIRij(sarea,skeyword,con);
        if (IRij == -1.0)
        {
         return 0;
        }
        /* Now, a call is made to compute sum for all areas in a
           particular keyword i */
        float SumIRij = SumAllIRij(skeyword,con);
        if (SumIRij == -1.0)
        {
         return 0;
        }
        /* Weight can now be computed */
```

```
            float Weightij;

            Weightij =  IRij* (float) Math.log(numberareas/SumIRij);

            /* Values are inserted into the table t_keywarea */
            if (sarea == null)
            {
             ps_1.setNull(1,Types.VARCHAR);
            }
            else
            {
             ps_1.setString(1,sarea);
            }
            skeyword.trim();
            ps_1.setString(2,skeyword);
            ps_1.setDouble(3,Weightij);
            ps_1.executeUpdate();

           } // End while
           s_1.close();
           ps_1.close();

         } // end try
        catch(SQLException exOb) {
          System.out.println("Error executing main body" +
exOb.getMessage());
          return 0;
        }
  return 1;
 } // End function


 public static float ComputeIRij(String area, String keyword,
Connection con)
 {
  int totalareakeyword,totalarea;
  ResultSet rs_1,rs_2;
  PreparedStatement ps_1,ps_2;

  /* First, the number of papers of area j in which keyword i occurs is
queried.
     If an error happens making any of the queries, a -1 is returned */

  String query1,query2;

  query1 = "SELECT COUNT(*) FROM STATIC WHERE AREA ";

  query2 = "SELECT COUNT(DISTINCT PID) FROM STATIC WHERE AREA ";

  try{
      if (area == null)
      {
       query1 = query1 + "IS NULL AND KEYWORD = ?";
       ps_1 = con.prepareStatement(query1);
       ps_1.setString(1,keyword);
      }
      else
```

```
      {
       query1 = query1 + "= ? AND KEYWORD = ?";
       ps_1 = con.prepareStatement(query1);
       ps_1.setString(1,area);
       ps_1.setString(2,keyword);
      }
      rs_1 = ps_1.executeQuery();
      rs_1.next();
      totalareakeyword = rs_1.getInt(1);

      /* The second query is executed, which refers to extracting total
         # of papers of areas j */

      if (area == null)
      {
       query2 = query2 + "IS NULL";
       ps_2 = con.prepareStatement(query2);
      }
      else
      {
       query2 = query2 + "= ?";
       ps_2 = con.prepareStatement(query2);
       ps_2.setString(1,area);
      }
      rs_2 = ps_2.executeQuery();
      rs_2.next();
      totalarea = rs_2.getInt(1);
      ps_1.close();
      ps_2.close();
     }
  catch(SQLException exOb){
      System.out.println("Error : "+ exOb.getMessage());
      return -1;
   }
  /* IRij is computed */

  float IRij = (float) totalareakeyword / totalarea;

  return IRij;
 }

 public static float SumAllIRij(String skeyword, Connection con)
 {
  String s_1 = "select area from static group by area,keyword having
keyword = ?";
  PreparedStatement ps_1;
  ResultSet rs_1;
  float sum = 0;

  try{
      ps_1 = con.prepareStatement(s_1);
      ps_1.setString(1,skeyword);
      rs_1 = ps_1.executeQuery(s_1);

      while(rs_1.next())
      {
        String sarea = rs_1.getString(1);
```

```
        float val = ComputeIRij(sarea,skeyword,con);
        if (val == -1.0)
         {
          return (float) -1.0;
         }
        sum += val;
      } // end while
      ps_1.close();
     } // end try
 catch(SQLException exOb){
   System.out.println("Error executing SumIRij :"+ exOb.getMessage());
   return 0;
 }
return sum;
} // end function

public static void main(String args[])
{
 int resp = ConstructRule();
 if (resp != 0 )
  {
   System.out.println("Program terminated successfully!");
  }
 else
  {
   System.out.println("Program abnormally terminaded!");
  }
} // end main.
}
```

Import_a.c

```
/*********************************************************************/
/* Program name :import_a.c                                          */
/* Made in : May 8, 1999                                             */
/* Description : This program import subject areas from a flat file. */
/* Subject areas are read from a text file and then imported into the*/
/* Hugin Network. The structure of the network would be : In the top */
/* level we will have the generic subject areas and then in the next */
/* level specific areas that are included into the any of the generic*/
/* areas and so on. From what it is seen from the text file, there   */
/* will be four different levels of classification of this subject   */
/* areas.                                                            */
/* The bottom level will hold the keywords that comes from the oracle*/
/* table source.                                                     */
/* Parameter:                                                        */
/*  1. Source file name. Source file where subject areas are stored  */
/*  2. new .hkb file. This file will contain the imported subject    */
/*     areas                                                         */
/*********************************************************************/

#include<hugin.h>
#include<string.h>
#include<ctype.h>
#define LINESIZE 101
#define ARGUMENTS 3

typedef struct
{
 char Name[LINESIZE]; /* Subject Name*/
 short int nspaces; /* amount of spaces to the left */
 char HNodeName[10]; /* Hugin assigned node name */
}Node;

Node * importsubjects(h_domain_t d_1, h_node_t parent,FILE * fd, Node
                                                              ant);
void error_message(const char * mes);
int getspace(char *);
int strcopy_range(char *, const char *,int,int);
void clearing_nonalpha(char *);

Node *importsubjects(h_domain_t d_1,h_node_t parent,FILE * fd, Node
                                                              ant)
{
 Node new_node,potential_node,*ptr_pot_node;
 char line[LINESIZE];
 h_node_t child;


 fgets(line, LINESIZE - 1, fd);
 if (!feof(fd))
 {
  new_node.nspaces = getspace(line);
  strcopy_range(new_node.Name,line,new_node.nspaces,strlen(line)-1);

  /* Creating the new node into the Hugin API */
```

```
child = h_domain_new_node(d_1,h_category_chance,h_kind_discrete);
if (child == NULL)
{
    printf("Child node could not be created!");
    return 0;
}
/* Code added on June 14, 1999 about taking out non-character symbols
      and taking also out the \n symbol from new_node.Name */

clearing_nonalpha(new_node.Name);
h_node_set_label(child, new_node.Name);
h_node_set_number_of_states(child,2);
h_node_set_state_label(child,1,"yes");
h_node_set_state_label(child,0,"no");

/* h_node_set_subtype(child, h_subtype_boolean);*/
/* assigned name is copied into the new_node structure */
strcpy(new_node.HNodeName,h_node_get_name(child));

while (1)
{
    if (new_node.nspaces > ant.nspaces)
    {
     /* Add the child to be linked to its parent */
     h_node_add_parent(child,parent);

     ptr_pot_node = importsubjects(d_1, child ,fd, new_node);
     /* We check if this is the end of file */
     if (!ptr_pot_node)
          break;
     /* Information is inmediatelly copied from the temporary
        variable */
     potential_node.nspaces = ptr_pot_node->nspaces;
     strcpy(potential_node.Name, ptr_pot_node->Name);
     strcpy(potential_node.HNodeName, ptr_pot_node->HNodeName);
     /* It is neccesary to check if new_node and potential_node are
        at the same level */
     if (potential_node.nspaces == new_node.nspaces)
     {
      /* potential node is at the same level as new_node, so they are
            siblings */
      child =
          h_domain_get_node_by_name(d_1,potential_node.HNodeName);
      new_node.nspaces = potential_node.nspaces;
      strcpy(new_node.Name,potential_node.Name);
      strcpy(new_node.HNodeName, potential_node.HNodeName);
     }
    else
     return (&potential_node);
    } /* end if (new_node.nspaces */
    else if (ant.nspaces >= new_node.nspaces)
                return &new_node;
 } /* end while */
} /* end if (!feof(fd)) */
return NULL;
}
```

```
int getspace(char * str_1)
{
 short int i;

 for(i=0; i < strlen(str_1) && str_1[i] == ' '; i++);
 return i;
}

/*********** function clearing_nonalpha*******************************/
/* This function takes out all those non-alphabetical characters from*/
/* the subjects                                                       */

void clearing_nonalpha(char * name)
{
 int i;
 if (!isalpha(name[0])) /* First letter is not a character */
  {
      /* The character must be erase, the string must be shifted */
      for(i=0;i < strlen(name) ; i++)
            name[i] = name[i+1];
  }
 /* We cut the last two characters */
 name[strlen(name)-1] = '\x0';
}


int strcopy_range(char * dest, const char * source, int init, int end)
{
 int i,c;
 for(i = init, c = 0; i <= end && i < strlen(source); i++,c++)
      dest[c] = source[i];
 dest[c] = '\0';
}


int main(int argc, char * argv[])
{
 FILE *fp,*fp1;
 char line[LINESIZE],i,netname[30];
 Node temp_1,*ptr_temp_1;
 h_domain_t d_categorization;
 h_node_t node_1;

 /*Opens the file for reading*/

 if (argc != ARGUMENTS)
 {
  error_message("Invalid # of arguments");
  exit(1);
 }

 /* The hugin domain is created */

 d_categorization = h_new_domain();

 if (!d_categorization)
 {
```

```
 printf("Domain could not be created");
 exit(1);
}


if ((fp = fopen(argv[1],"r")) == NULL)
{
 error_message("File could not be opened");
 exit(1);
}

fgets(line,LINESIZE - 1,fp);
temp_1.nspaces = getspace(line);
strcopy_range(temp_1.Name,line,temp_1.nspaces -1, strlen(line));

/* Node is created in this line */
node_1 = h_domain_new_node(d_categorization, h_category_chance,
h_kind_discrete);
if (node_1 == NULL)
{
 printf("Node %s could not be created!",temp_1.Name);
 exit(1);
}
clearing_nonalpha(temp_1.Name);
h_node_set_label(node_1, temp_1.Name);
h_node_set_number_of_states(node_1,2);
h_node_set_state_label(node_1,1,"yes");
h_node_set_state_label(node_1,0,"no");
strcpy(temp_1.HNodeName,h_node_get_name(node_1));

if (feof(fp))
{
 error_message("Cannot read File");
 exit(1);
}
while(!feof(fp))
 {
     ptr_temp_1 = importsubjects(d_categorization,node_1,fp,temp_1);
     if (ptr_temp_1)
     {
      temp_1.nspaces = ptr_temp_1->nspaces;
      strcpy(temp_1.Name,ptr_temp_1->Name);
      strcpy(temp_1.HNodeName,ptr_temp_1->HNodeName);
      node_1 =
      h_domain_get_node_by_name(d_categorization,temp_1.HNodeName);
     }
} /* End while */
 close(fp);

 /* Saving the domain in two different formats */
 h_domain_save(d_categorization, argv[2], h_endian_host);

 /* Saving the domain using NET language */
 strncpy(netname,argv[2],25);
 strcat(netname,".net");
 fp1 = fopen(netname,"w");
 if (fp1)
```

```
        h_domain_write_net(d_categorization,fp1);
 return 1;
}

void error_message(const char * mes)
{
 printf("%s\n",mes);
}
```

Buildprob.c

```
/*********************************************************************/
/* Program name :buildprob.c                                         */
/* Made in : June 5, 1999                                            */
/* Description : This program assigns probabilities to areas nodes.  */
/* Probabilities are computed as follows.                            */
/* If node does not have parents, we assume an equal probability of  */
/* 0.5 for each case. In the case of node with parents; I count the  */
/* number of siblings and 1/# of siblings is the probability for each*/
/* node.                                                             */
/* Parameters : Just the name of the .hkb file                       */
/*********************************************************************/

#include<hugin.h>
#include<stdlib.h>
#include<string.h>
#define LINESIZE 101
#define ARGS 2

/************************ struct declaration ************************/

enum Optionprob {CHECKED, UNCHECKED};
struct NodeCheck
{
 char Name[6]; /* Node name */
 enum Optionprob has_prob; /* Has probability */
 short int level; /* level of the node in the bayesian network */
 };

typedef struct
{
 int n;
 struct NodeCheck *p;
}ArrayCheck;

/************************function declaration*************************/
int count_nodes(h_domain_t d);
void extract_nodename(ArrayCheck *array_p);
void FindFirstUncheckedNode(ArrayCheck * array_ptr, char * name);
void check_node(ArrayCheck * array_1, h_node_t node_1, short int
level);
int save_categories(ArrayCheck array_1);
void AssignProbabilities(h_node_t node_1, short int lev, int nsiblings,
                                                  ArrayCheck * array_1);


int count_nodes(h_domain_t d)
{
 h_node_t n;
 int count = 0;

 for(n = h_domain_get_first_node(d); n != 0; n = h_node_get_next(n))
  count++;
 return count;
}
```

```
void extract_nodename(ArrayCheck *array_p)
{
 int i;
 char nname[6];
 for(i = 0; i < array_p->n; i++)
 {
  sprintf(nname,"C%d",i+1);
  strcpy(array_p->p[i].Name,nname);
  array_p->p[i].has_prob = UNCHECKED;
  array_p->p[i].level = 0;
 }
}

void FindFirstUncheckedNode(ArrayCheck * array_ptr, char * name)
{
 int i = 0;
 while( array_ptr->p[i].has_prob == CHECKED)
             i++;
 if ( i == array_ptr->n)
      strcpy(name,"\0");
 else
      strcpy(name,array_ptr->p[i].Name);
}

void check_node(ArrayCheck * array_1, h_node_t node_1, short int level)
{
 char name[LINESIZE];
 int i=0;

 strcpy(name, h_node_get_name(node_1));
 for(i = 0; i < array_1->n; i++)
 {
  if (!strcmp(array_1->p[i].Name,name))
       {
        array_1->p[i].has_prob = CHECKED;
        array_1->p[i].level = level;
        break;
       }
 } /* end for */
} /* end function */

int save_categories(ArrayCheck array_1)
{
 FILE *fd;
 char line[LINESIZE];
 int i = 0;

 fd = fopen("level.dat","w");
 if (fd == NULL)
 {
  fprintf(stderr," Error opening level.dat!");
  return 0;
 }

 for(i = 0; i < array_1.n; i++)
```

```
 {
  sprintf(line,"%s,%d\n",array_1.p[i].Name,array_1.p[i].level);
  fputs(line,fd);
 } /* end for */
 close(fd);
} /* End function */

void AssignProbabilities(h_node_t node_1, short int lev, int nsiblings,
                                                          ArrayCheck
* array_1)
{
 int i;
 float prob;
 h_table_t table_1;
 h_number_t *array_val;
 h_node_t *node_list,*p;
 char slevel[5];

 table_1 = h_node_get_table(node_1);
 if (table_1 == NULL)
  {
      printf("Error reading table\n");
      return;
  }
 array_val = h_table_get_data(table_1);
 if (array_val == NULL)
 {
  printf("Error getting values of the table\n");
  return;
 }

 /* The level of the node is checked, to see if it's a root node or a
    linked node */
 if (lev == 0) /* If node is root */
  {
      /* This node has only two probabilities :
            Prob of being yes and Prob. of being No */
      array_val[0] = 0.5; /* In this case prob. are assigned equally */
      array_val[1] = 0.5;
 }
 else
  {
      prob = (float) 1/nsiblings;
      array_val[0] = 1; /* A = No | B = No */
      array_val[1] = 0; /* A = Yes | B = No; imposible for our case */
      array_val[2] =  (float) 1 - prob; /* A = No | B = yes */
      array_val[3] = prob; /* A = yes | B = yes; this is the prob of
                   a document being categorized in an specific
                   subarea, given that the document is already
                   categorized in a subject */
  }
 sprintf(slevel,"%d",lev);
 h_node_set_attribute(node_1,"level",slevel);

 check_node(array_1, node_1, lev); /* node is marked as visited */
 /* The children of node_1 are visited from left to right */
 node_list = h_node_get_children(node_1);
```

```
 if (node_list != NULL)
 {
  /* Get the count of number of children */
  i = 0;
  for(p = node_list; *p != 0; p++)
            i++;
  nsiblings = i;
  lev++; /* Next level */
  p = node_list;
  while (*p != 0)
  {
      AssignProbabilities((*p), lev, nsiblings, array_1);
      p++;
  }
 } /* End if (node_list ...*/
} /* End AssignProb ... */

int main(int argc, char * argv[])
{
 ArrayCheck array_1;
 h_domain_t d;
 h_node_t node_1;
 int n_domain,nsiblings;
 short int level;
 char Nodename[LINESIZE],filename[30];
 FILE *fp;

 if (argc != ARGS)
  {
      fprintf(stderr,"Invalid number of arguments");
      exit(EXIT_FAILURE);
  }
 if ((d = h_load_domain(argv[1])) == NULL)
 {
  fprintf(stderr,"h_load_domain failed : %s\n",
                  h_error_description(h_error_code()));
  exit(EXIT_FAILURE);
 }
 n_domain = count_nodes(d);
 array_1.p = (struct NodeCheck *) calloc(sizeof(struct
                                          NodeCheck),n_domain);
 array_1.n = n_domain;
 extract_nodename(&array_1);
 do{
       FindFirstUncheckedNode(&array_1,Nodename);
       if (strcmp(Nodename,"\0"))
       {
        node_1 = h_domain_get_node_by_name(d,Nodename);
        level = 0;
        nsiblings = 0;
        if (node_1 != 0)
              AssignProbabilities(node_1, level, nsiblings, &array_1);
       }
  }while(strcmp(Nodename,"\0") && node_1 != 0);
 free(array_1.p);
 if ((h_domain_save(d,argv[1],h_endian_host)) != 0)
  {
```

```
        fprintf(stderr,"Error saving the domain");
        exit(EXIT_FAILURE);
    }

   /* Saving the domain in NET language*/
   sprintf(filename,"%s.net",argv[1]);
   if ((fp = fopen(filename,"w")) == NULL)
   {
    fprintf(stderr,"Error saving the domain in NET language");
    exit(EXIT_FAILURE);
   }
   if ((h_domain_write_net(d,fp)) != 0)
   {
    fprintf(stderr,"Error writing the net file : %s",h_error_description(

                    h_error_code()));
    exit(EXIT_FAILURE);
   }
   close(fp);
   printf("Program terminated successfully!\n");
}
```

Impkey.pc

```
/*************************************************************************/
/* Program name : drawnet.c                                              */
/* Date : June 19, 1999                                                  */
/* Description : This programs imports keywords from an oracle table.*/
/* The oracle table is called weight, which is generated by the rule */
/* constructor program, and is used to store the value that          */
/* represents the relationship between keyword and subject areas.    */
/* These values as well as the relationship are imported into the    */
/* Hugin as user defined fields in each of the keywords that are     */
/* imported.                                                         */
/* When keywords are imported, new nodes are created and the         */
/* relationship that are represented in the oracle table "weight"    */
/* are also created as links to the actual subjects already stored   */
/* in the .hkb file                                                  */
/* Parameters :                                                      */
/* 1. Treshold : based value from where we start to import keywords. */
/*               Keyword-Areas under this value will not be added.   */
/* 2. Oracle User name : Username account in order to access Oracle. */
/* 3. Oracle Password : Password for Oracle.                         */
/* 4. Hugin .hkb file : the .hkb file with the current network, which*/
/*                      at this point should only include subjects.  */
/*                                                                   */
/*************************************************************************/

#include<sqlca.h>
#include<hugin.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#include<errno.h>

#define LINESIZE 101
#define ARGS 5

void sql_error(char * mesg);
h_node_t selectbycriteria(h_domain_t d_1, int (*f)(const char
*name,h_node_t node_1), char * name_1);
int find_node(const char * name, h_node_t n_1);
void uppercase(char * st_1);
void delete_spaces(char * name);

/* Template function to select a node according to a particular
   criteria function*/

h_node_t selectbycriteria(h_domain_t d_1, int (*f)(const char
*name,h_node_t node_1), char * name_1)
{
 short int flag = 0;
 h_node_t n_1;

 n_1 = h_domain_get_first_node(d_1);

 while (n_1 && !flag)
 {
```

```
      if ((*f)(name_1, n_1))
            flag = 1;
      else
            n_1 = h_node_get_next(n_1);
 }
 if (flag)
 {
  return (n_1);
 }
 else
      return 0;
}

/****** This function extract spaces from a string *************/

void delete_spaces(char * name)
{
 int i=0,j;
 while(name[i] == ' ' && name[i] != '\0')
 {
  if (name[i] == ' ')
  {
      /* Shift the characters to the left */
      for(j = i; j < strlen(name) - 1; j++)
       name[j]=name[j+1];
  }
  else
       i++;
 } /* End while */
/* From the end of the string, I go backward to eliminate unnecessary
      spaces */
 i = strlen(name)-1;
 while(name[i] == ' ' && i > 0)
 {
  name[i] = '\0';
  i--;
 }/* End while */

} /* End function */

/**This function converts a string into an uppercase string *****/

void uppercase(char * st_1)
{
 int i;
 for(i = 0; i < strlen(st_1); i++)
       st_1[i] = toupper(st_1[i]);
}

/*  This functions finds a node with a particular label */

int find_node(const char * name, h_node_t n_1)
{
 char NodeName[LINESIZE],name_1[LINESIZE];

 strcpy(NodeName, h_node_get_label(n_1));
 strcpy(name_1,name);
```

```
 uppercase(NodeName);
 uppercase(name_1);
 if (!strcmp(name_1,NodeName))
      return 1;
 return 0;
}


void sql_error(char * msg)
{
 char err_msg[128];
 int buf_len, msg_len;

 EXEC SQL WHENEVER SQLERROR CONTINUE;

 printf("\n%s\n",msg);
 buf_len = sizeof(err_msg);
 sqlglm(err_msg, &buf_len, &msg_len);
 printf("%.*s\n",msg_len, err_msg);
 EXEC SQL ROLLBACK RELEASE;

 exit(1);
}


int main(int argc, char * argv[])
{
 char keyw_1[50],area_1[LINESIZE],netname[35];
 float treshold;
 FILE *fd;
 char username[30],password[30],domain_name[30];
 char field_name[30],sweight[LINESIZE];

 h_domain_t d_1;
 h_node_t n_1,n_2;

 EXEC SQL DECLARE c_keywordarea CURSOR FOR
 SELECT KEYWORD, AREA, TO_CHAR(WEIGHT)
 FROM WEIGHT
 WHERE WEIGHT > :treshold
 ORDER BY KEYWORD,AREA;

 if (argc != ARGS)
 {
      fprintf(stderr,"Invalid number of arguments");
      exit(EXIT_FAILURE);
 }

 EXEC SQL WHENEVER SQLERROR DO sql_error("ORACLE error--\n");

 strcpy(username,argv[2]);
 strcpy(password,argv[3]);
 treshold = atof(argv[1]);
 if (errno == ERANGE)
 {
  perror("Error reading weight values from Hugin");
  exit(EXIT_FAILURE);
```

```
}

EXEC SQL CONNECT :username IDENTIFIED BY :password;

/* Copying the domain name */

strcpy(domain_name,argv[4]);
if ((d_1 = h_load_domain(domain_name)) == NULL)
{
 fprintf(stderr,"h_load_domain failed : %s\n",
          h_error_description(h_error_code()));
 exit(EXIT_FAILURE);
}
/* The cursor is open */
EXEC SQL OPEN c_keywordarea;

EXEC SQL WHENEVER NOT FOUND DO break;

for(;;)
{
 EXEC SQL FETCH c_keywordarea INTO :keyw_1, :area_1, :sweight;
 delete_spaces(area_1);
 delete_spaces(keyw_1);

 n_1 = selectbycriteria(d_1,find_node,keyw_1);
 if (n_1 == 0) /* New keyword node */
 {
      n_1 = h_domain_new_node(d_1, h_category_chance,
                                        h_kind_discrete);
      h_node_set_label(n_1, keyw_1);
      h_node_set_number_of_states(n_1, 2);
      h_node_set_state_label(n_1, 0, "no");
      h_node_set_state_label(n_1, 1, "yes");
      h_node_set_attribute(n_1,"cat_type","K");
 } /* end if (n_1... */
 /* We look for the node corresponding to this area */
 n_2 = selectbycriteria(d_1, find_node, area_1);
 if (n_2 == 0) /* If subject area does not exist */
 {
     /* The keyword node must be deleted since it does not have
          parents */
     fprintf(stderr,"Subject area: %s does not exist\n", area_1);
     h_node_delete(n_1);
 } /* end if (n_2 */
 else
      {
       /* Keyword and Subject nodes are connected */
       h_node_add_parent(n_1,n_2);
       sprintf(field_name,"cat_%s_type",h_node_get_name(n_2));
       h_node_set_attribute(n_1,field_name,sweight);
      }
} /* End for */
EXEC SQL CLOSE c_keywordarea;
if (h_domain_save(d_1, domain_name, h_endian_host) != 0)
{
 fprintf(stderr, "Error saving the domain %s!",domain_name);
 exit(EXIT_FAILURE);
```

```
} /* end if (h_domain_save */
sprintf(netname,"%s.net",domain_name);
fd = fopen(netname,"w");
if (fd == NULL)
{
 fprintf(stderr, "Error writing to file: %s", netname);
 exit(EXIT_FAILURE);
}
if (h_domain_write_net(d_1, fd) != 0)
{
 fprintf(stderr,"Error saving domain");
 exit(EXIT_FAILURE);
}
close(fd);
printf("Program terminated successfully!\n");
} /* End main */
```

Assignkp.c

```
/*********************************************************************/
/* Program name :assignkp.c                                          */
/* Made in : July 5, 1999                                            */
/* Description : This program assigns probabilities to keyword nodes.*/
/* Probabilities are based on weight values of the relationship      */
/* between keywords and areas, taken from the rule constructor       */
/* program.                                                          */
/* The probability works this way : For each keyword node,           */
/* attached is                                                       */
/* the weight values of its parents. These values are first added to */
/* find the "total weight" value, and then the probability for each  */
/* pair keyword node - area node is the weight value of that         */
/* relationship stored in the keyword node over the "total weight".  */
/* For Those cases where there is only one parent for a keyword node;*/
/* I assumed a value of 0.3 for no-yes, which means : "Probability of*/
/* this particular keyword not being present, given that the subject */
/* area is present ". In the other case I assummed a value of 0.7.   */
/* Parameter: the name of the Hugin Knowledge Base file.             */
/*********************************************************************/

#include<hugin.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<errno.h>

#define LINESIZE 101
#define MAX_PARENTS_PER_NODE 35
#define ARGS 2


int h_node_get_nparents(h_node_t n_1);
void get_binarystring(unsigned int val, char * st_1,unsigned int
max_size);
void ComputeKeywordProb(h_node_t node_1);

int main(int argc, char * argv[])
{
 h_domain_t d;
 h_node_t node_1;
 h_string_t p;
 char filename[LINESIZE];
 FILE *fp;

 if (argc != ARGS)
  {
      fprintf(stderr,"Invalid number of arguments");
      exit(EXIT_FAILURE);
  }
 if ((d = h_load_domain(argv[1])) == NULL)
 {
  fprintf(stderr,"h_load_domain failed : %s\n",
                        h_error_description(h_error_code()));
  exit(EXIT_FAILURE);
```

```
}

/* Network traversing */
for(node_1 = h_domain_get_first_node(d); node_1 != 0;
           node_1 = h_node_get_next(node_1))
{
 p = h_node_get_attribute(node_1,"cat_type");
 if (p != NULL) /* If p is not null, then node is a keyword node */
      ComputeKeywordProb(node_1);
}  /* End for */

/* Saving the domain */
if ((h_domain_save(d,argv[1],h_endian_host)) != 0)
 {
     fprintf(stderr,"Error saving the domain");
     exit(EXIT_FAILURE);
 }

/* Saving the domain in NET language*/
sprintf(filename,"%s.net",argv[1]);
if ((fp = fopen(filename,"w")) == NULL)
{
 fprintf(stderr,"Error saving the domain in NET language");
 exit(EXIT_FAILURE);
}
if ((h_domain_write_net(d,fp)) != 0)
{
 fprintf(stderr,"Error writing the net file : %s",h_error_description(

                 h_error_code()));
 exit(EXIT_FAILURE);
}
close(fp);
printf("Program terminated successfully!\n");
}

/* This function computes the probability for each keyword. Probability
   is based in the weight values of the keyword parents.              */

void ComputeKeywordProb(h_node_t node_1)
{
 h_node_t *p,*parents;
 h_table_t table_1;
 h_number_t *array_val;
 int nparents;
 int i,j; /* Traverse the index */
 float numerator,denominator; /* used in computing probabilities */
 float prob; /* Probability value */
 float sum; /* Sum of all weight of all parents nodes */
 h_string_t name,svalue;
 char sname[MAX_PARENTS_PER_NODE],stbinary[MAX_PARENTS_PER_NODE];
 float weight; /* weight of the node */
 size_t tsize; /* Table size */

 parents = h_node_get_parents(node_1);
 if (*parents != 0)
 {
```

```
/* We first compute the sum of all weight of all parents */
sum = 0;
for (p = parents; *p != 0; p++)
{
 /* We get the node name */
 name = h_node_get_name(*p);
 /* name is concatenated with cat_<name>_type */
 sprintf(sname,"cat_%s_type",name);
 svalue = h_node_get_attribute(node_1,sname);
 weight = atof(svalue);
 if (errno == ERANGE)
 {
      perror("Error reading weight values from Hugin");
 }
 sum += weight;
} /* for (p = */

 /* Table is required */
table_1 = h_node_get_table(node_1);
if (table_1 == NULL)
{
 printf("Error reading table");
 return;
}
/* Index is required */
array_val = h_table_get_data(table_1);
if (array_val == NULL)
{
 printf("Error getting values of the table");
 return;
}

/** If number of parents of node is less or equal than one, then
    there is no need to make any computes. A value is assumed for
    each of the four cases */

nparents = h_node_get_nparents(node_1);

if (nparents == 1)
{
 array_val[0] = 1;
 array_val[1] = 0;
 array_val[2] = (float) 0.3;
 array_val[3] = (float) 0.7;
} /* end if (nparents ...*/
else /*Rest of the code ****/
{
 /* default values are assigned to the first two cases, since
   they are always the same */
 array_val[0] = 1;
 array_val[1] = 0;

 /* Size of the table is required */
 tsize = h_table_get_size(table_1);
 if (tsize == -1)
  {
      fprintf(stderr,"error reading the size of the network\n");
```

```
        }
       /* Assignment of probabilities to the rest of nodes */
       for(i = 2; i < tsize; i += 2)
        {
            /* The index is converted in binary format */
            get_binarystring(i,stbinary,nparents + 1);
            j = numerator = denominator = 0;
            for(p = h_table_get_nodes(table_1); *p != 0; p++)
             {
              if (stbinary[j] == '1') /*Node p is involved in this case
                                        and the value*/
                  {                    /*  is required for computing
                                        probability */
                   name = h_node_get_name(*p);
                   /* name is concatenated with cat_<name>_type */
                   sprintf(sname,"cat_%s_type",name);
                   svalue = h_node_get_attribute(node_1,sname);
                   weight = atof(svalue);
                   if (errno == ERANGE)
                    {
                     perror("Error reading weight values from Hugin");
                    }
                   if (numerator == 0)
                         numerator = weight;
                   else
                         numerator *= weight;
                   if (denominator == 0)
                          denominator = sum;
                   else
                          denominator *= sum;
                  } /* end if (stbinary[j] */
               j++;
              } /* end for (p = ...*/
              prob = (float) numerator/denominator;
              array_val[i] = (float) 1 - prob;
              array_val[i+1] = (float) prob;
         } /* End for (i = 2.. */
       } /* end else rest of code */
 } /* if ((parents ... */
else  /* Keyword node is deleted since it does not have any parents */
 if (h_node_delete(node_1))
  {
       fprintf(stderr,"h_node_delete failed : %s\n",
       h_error_description(h_error_code()));
       exit(EXIT_FAILURE);
  }
} /* void Compute... */


/* This function counts the number of parents of a node */

int h_node_get_nparents(h_node_t n_1)
{
 h_node_t *p;
 int n = 0;
 for(p = h_node_get_parents(n_1); *p != 0; p++)
        n++;
```

```
 return n;
}


/* This function converts an integer value into an string in binary
format */

void get_binarystring(unsigned int val, char * st_1,unsigned int
max_size)
{
 int i,temp_val;

 i = max_size - 1;
 while( val > 0 && i >= 0)
 {
      temp_val = val & 1;
      if (temp_val)
            st_1[i] = '1';
      else
            st_1[i] = '0';
      i--;
      val = val >> 1;
 } /* End while */
 if (i != 0)
 {
  for(;i >= 0;i--)
            st_1[i] = '0';
 }
}
```

Drawnet.c

```
/*********************************************************************/
/* Program name : drawnet.c                                          */
/* Date : July 13, 1999                                              */
/* Description : This programs generates the graphical coordinates in*/
/* order for the categorization network to be displayed in the Hugin */
/* Runtime module.                                                   */
/* This program assumes that no level has been assigned for the      */
/* keywords.                                                         */
/* So what it does, it assignes the lower level to keyword, so       */
/* keywords will be shown as leaves of the network.                  */
/* If keywords has been assigned a level, then the program,          */
/* recalculates positions and sets the new positions for every node. */
/* Parameters :                                                      */
/*    1. .hkb file with the corresponding categorization network     */
/*    2.  Width of the node.                                         */
/*    3.  Height of the node.                                        */
/*********************************************************************/

#include<hugin.h>
#include<stdlib.h>
#include<string.h>


#define LINESIZE 101
#define MAX_PARENTS_PER_NODE 35
#define ARGS 4
#define INIT_X_AX 10
#define INIT_Y_AX 5
#define VERTICAL_SPACE 15
#define HORIZONTAL_SPACE 5
#define MAXLEVELS 6

int FindMaxLevel(h_domain_t d);
void SetKeywordLevel(int level,h_domain_t d);


typedef struct
{
 int level;
 int current_x;
 int current_y;
}PosLevel;

/* This functions sets the level to the keywords in order for them to
   be displayed graphically.
   If no level has been set for keywords, this programs assumes that
   keywords are located at the lower level of the network(Highest level
   value). */

void SetKeywordLevel(int level,h_domain_t d)
{
 h_node_t node_1;
 h_string_t p,p1;
 char slevel[10];
```

```
 sprintf(slevel,"%d",level);

  /* Network traversing */
 for(node_1 = h_domain_get_first_node(d); node_1 != 0;
            node_1 = h_node_get_next(node_1))
 {
  p = h_node_get_attribute(node_1,"cat_type");
  if (p != NULL) /* The node is a keyword node */
      {
        p1 = h_node_get_attribute(node_1,"level");
        if (p1 == NULL)    /* If level field is not set, then we added
                              it */
            {
             h_node_set_attribute(node_1,"level",slevel);
            }
      }
 }  /* End for */
} /* End function */

/* This functions finds the maximum level value already set in the
network */

int FindMaxLevel(h_domain_t d)
{
 int maxlevel,level_1;
 h_string_t p;
 h_node_t n_1;

 maxlevel = 0;
 for(n_1 = h_domain_get_first_node(d); n_1 != 0; n_1 =
                                          h_node_get_next(n_1))
 {
  p = h_node_get_attribute(n_1,"level");
  if (p != NULL)
  {
      level_1 = atoi(p);
      if (maxlevel < level_1)
                  maxlevel = level_1;
  }
 }
 return maxlevel;
}

/*                Main Function                                    */

int main(int argc, char * argv[])
{
 h_domain_t d;
 h_node_t node_1;
 h_node_t *p;
 char filename[LINESIZE];
 FILE *fp;
 h_string_t is_checked,slevel;
 char slevel_string[3];
 int width,height,mlevel,i,level;
 h_coordinate_t x,y;
```

```
PosLevel current_coordinates[MAXLEVELS];

if (argc != ARGS)
 {
     fprintf(stderr,"Invalid number of arguments");
     exit(EXIT_FAILURE);
 }
if ((d = h_load_domain(argv[1])) == NULL)
{
 fprintf(stderr,"h_load_domain failed : %s\n",
h_error_description(h_error_code()));
 exit(EXIT_FAILURE);
}
/* We get the width and the height value */
width = atoi(argv[2]);
height = atoi(argv[3]);

/* Value returned by atoi is checked */
if (width == 0 || height == 0)
{
 fprintf(stderr,"Numeric values are not valid for width and height
                                                              \n");
 exit(EXIT_FAILURE);
}

/* The node size is defined */
h_domain_set_node_size(d,width,height);

/* We look for the highest level, in order to assign highest level + 1
   to keyword nodes */

mlevel = FindMaxLevel(d);


/* We assigned mlevel + 1 to keywords */

mlevel++;

SetKeywordLevel(mlevel,d);

for(i = mlevel; i > 0 && mlevel < MAXLEVELS; i--)
{
 current_coordinates[i].level = i ;
 if ((i % 2) == 0)
     {
      current_coordinates[i].current_x = INIT_X_AX;
     }
 else
      current_coordinates[i].current_x = 2*INIT_X_AX;
 current_coordinates[i].current_y = INIT_Y_AX + (mlevel -
i)*(VERTICAL_SPACE + height);
}

/* Network traversing */
for(node_1 = h_domain_get_first_node(d); node_1 != 0;
           node_1 = h_node_get_next(node_1))
{
```

```
 is_checked = h_node_get_attribute(node_1,"checked");
 if (is_checked == NULL)
     {
      slevel = h_node_get_attribute(node_1,"level");
      level = atoi(slevel);
      x = current_coordinates[level].current_x;
      y = current_coordinates[level].current_y;
      h_node_set_position(node_1,x,y);
      /* Increment current_coordinates_x according to the node size
         and the space */
      current_coordinates[level].current_x += (width +
                                               HORIZONTAL_SPACE);
      h_node_set_attribute(node_1,"checked","y");
     } /* end if (!strcmp ... */
 /* We evaluate the parents of this node */
 for( p = h_node_get_parents(node_1); *p != 0 ; p++)
 {
     is_checked = h_node_get_attribute(*p, "checked");
     if (is_checked == NULL) /* Node is not checked */
     {
      /* We get the position for the level at this node */
      slevel = h_node_get_attribute(*p,"level");
      level = atoi(slevel);
      x = current_coordinates[level].current_x;
      y = current_coordinates[level].current_y;
      h_node_set_position(*p, x , y);
      current_coordinates[level - 1].current_x +=  (width +
                                               HORIZONTAL_SPACE);
      h_node_set_attribute(*p,"checked","y");
     } /* End if it is checked */
 } /* End for */
} /* End for (node_1 ... */

/* The domain is compiled */

if (!h_domain_is_compiled(d))   /* If domain is not compiled */
{
 if ( h_domain_compile(d) != 0)
     {
      fprintf(stderr,"h_domain_compile failed: %s\n",

h_error_description(h_error_code()));
     }
}

/* Saving the domain */
if ((h_domain_save(d,argv[1],h_endian_host)) != 0)
 {
     fprintf(stderr,"Error saving the domain");
     exit(EXIT_FAILURE);
 }

/* Saving the domain in NET language*/
sprintf(filename,"%s.net",argv[1]);
if ((fp = fopen(filename,"w")) == NULL)
{
 fprintf(stderr,"Error saving the domain in NET language");
```

```
  exit(EXIT_FAILURE);
 }
 if ((h_domain_write_net(d,fp)) != 0)
 {
  fprintf(stderr,"Error writing the net file : %s",h_error_description(
                                                    h_error_code()));
  exit(EXIT_FAILURE);
 }
 close(fp);
 printf("Program terminated successfully!\n");
}
```

APENDIX C

REVIEWER SELECTION SMALL EXAMPLE DATA

Here is a detailed specification of input information to this problem : first, you will see the input ASCII file corresponding to the different AI subjects, then the conditional probabilities for all the nodes that compound the example of Figure 12.

```
AI
        distributed AI
                decision trees
                        real-time systems
                        search
                planning
                        search
                        causality
        description logic
                planning
                        search
                        causality
                theorem proving
                        causality
          cognitive modeling
```

Figure 18 Ontology File

AREA	KEYWORD	WEIGHT
real-time systems	real-time	4.7
distributed AI	real-time	6.89
decision trees	simplex	5.0
description logic	pathways	4.2
planning	membership	3.69
causality	membership	4.59
theorem proving	epsilon	4.30
cognitive modeling	epsilon	4.49

Figure 19 Weight Table

Conditional Probability Table 1 real-time systems.

State	Value
Yes	0.25
No	0.75

Conditional Probability Table 2 search.

State	Value
Yes	0.25
No	0.75

Conditional Probability Table 3 causality.

State	Value
Yes	0.25
No	0.75

Conditional Probability Table 4 Cognitive modeling.

State	Value
Yes	0.25
No	0.75

Conditional Probability Table 5 decision trees.

search	False | No		True | Yes	
Real-time sys.	False|No	True|Yes	False|No	True|Yes
False|No	1	0	0	0
True|Yes	0	1	1	1

Conditional Probability Table 6 planning

causality	False | No		True | Yes	
search	False|No	True|Yes	False|No	True|Yes
False|No	1	0	0	0
True|Yes	0	1	1	1

Conditional Probability Table 7 theorem proving

Cognit. Mod.	False | No		True | Yes	
causality	False|No	True|Yes	False|No	True|Yes
False|No	1	0	0	0
True|Yes	0	1	1	1

Conditional Probability Table 8 Distributed AI

planning	False | No		True | Yes	
Decision trees	**False|No**	**True|Yes**	**False|No**	**True|Yes**
False|No	1	0	0	0
True|Yes	0	1	1	1

Conditional Probability Table 9 Description logic

Theor. Prov.	False | No		True | Yes	
planning	**False|No**	**True|Yes**	**False|No**	**True|Yes**
False|No	1	0	0	0
True|Yes	0	1	1	1

Conditional Probability Table 10 AI

Desc. log	False | No		True | Yes	
Dist. AI	**False|No**	**True|Yes**	**False|No**	**True|Yes**
False|No	1	0	0	0
True|Yes	0	1	1	1

Conditional Probability Table 11 simplex

Decis. Trees	False|No	True|Yes
False|No	0.6	0.45
True|Yes	0.4	0.55

Conditional Probability Table 12 pathways

Desc. Logic	False|No	True|Yes
False|No	0.7	0.25
True|Yes	0.3	0.75

Conditional Probability Table 13 real-time

Dist. AI.	False | No		True | Yes	
Real-time sys.	False|No	True|Yes	False|No	True|Yes
False|No	0.75	0.35	0.41	0.37
True|Yes	0.25	0.65	0.59	0.63

Conditional Probability Table 14 membership

Causality	False | No		True | Yes	
planning	**False|No**	**True|Yes**	**False|No**	**True|Yes**
False|No	0.9	0.2	0.35	0.26
True|Yes	0.1	0.8	0.65	0.74

Conditional Probability Table 15 epsilon

Cog. Mod.	False | No		True | Yes	
Theo. proving	**False|No**	**True|Yes**	**False|No**	**True|Yes**
False|No	0.85	0.42	0.59	0.51
True|Yes	0.15	0.58	0.41	0.49

REFERENCES

1. E. Charniak, "Bayesian Networks without Tears", *AI MAGAZINE*, American Association for Artificial Intelligence (AAAI)., Menlo Park, CA, 1991, pp. 50-64.

2. J. Geller, "Challenge: How IJCAI 1999 can prove the value of AI by using AI", *Proc. Of the Fifteenths International Joint Conference on Artificial Intelligence (IJCAI 1997)*, Morgan Kaufmann Publishers, Nagoya, Japan, August 1997, pp. 55-58.

3. F. Jensen, *An Introduction to Bayesian Networks*, Springer-Verlag, New York, NY, 1996.

4. G. Salton, *Automatic Text Processing, The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley Publishing Co., Reading, MA, 1988, pp. 279-280.

5. G. Weiderhold. "Model-free optimization.", *Proc. DARPA Software Technology Conference*, Meridien Corp. Arlington, VA, 1992, pp. 83-96.