2008

# A software system for causal reasoning in causal Bayesian networks

Lexin Liu
*Iowa State University*

A software system for causal reasoning in causal Bayesian networks

by

**Lexin Liu**

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Jin Tian, Major Professor
Doug Jacobson
Shashi Gadia

Iowa State University

Ames, Iowa

2008

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

Knowing the cause and effect is important to researchers who are interested in modeling the effects of actions, and Artificial Intelligence researchers are among them. One commonly used method for modeling cause and effect is graphical model. Bayesian Network is a probabilistic graphical model for representing and reasoning uncertain knowledge. It has been used as a fundamental tool and is becoming a more and more important area for research and application in the AI field. A common graphical causal model used by many researchers in AI field is a directed acyclic graph (DAG) with causal interpretation known as the causal Bayesian network (BN). Causal reasoning and causal understanding are the causal interpretation part of a causal Bayesian Network. They enable people to find meaningful order in events that might otherwise appear random and chaotic. Further more, they can even help people to plan and predict the future. In this thesis, we develop a software system, which is a set of tools to solve causal reasoning problems, such as to identify unconditional causal effects, to identify conditional causal effects and to find constraints in a causal Bayesian Networks with hidden variables. The features of the software system are presented in detail and the applications of the software system are discussed.

# CHAPTER 1. INTRODUCTION

In order to model the effects of actions, it is crucial for the researchers knowing the cause and effect. One famous example would be the relationship of smoking and lung cancer. The observation of a statistical correlation between smoking and lung cancer alone can not lead to the conclusion that the cessation of smoking will change one's chances of getting lung cancer. However, if smoking is a cause for lung cancer, then the conclusion of one's choice to continue or quit smoking will affect one's chances of getting lung cancer is valid [1].

Researchers in Artificial Intelligence (AI) field, among others, are interested in identifying causal effect relationships in order to modeling the effects of actions. Graphical model is great for modeling cause and effect because it is intuitionistic and easy to be understood. It is not a surprise that causal reasoning with graphical models has been a hot topic in the artificial intelligence community in recent years [1]. Bayesian network (BN) has been used as a fundamental tool for the representation and manipulation of beliefs. It is a probabilistic graphical model for representing and reasoning uncertain knowledge and is becoming a more and more important area for research and application in the AI field [2, 3]. Causal Bayesian network, a directed acyclic graph (DAG) with causal interpretation, is a common graphical causal model used by many researchers in AI field [4].

The purpose of this thesis is to develop a software system, which is a set of tools to create and manipulate causal Bayesian networks. In the mean time, the software system can identify causal effects from non-experimental data in a causal Bayesian network and it can also find independence and non-independence constraints from a causal Bayesian network.

## 1.1 Bayesian Network [BN]

In nineteen eighties, there was resurgence and new acceptance of the probability and decision theory in AI community after Peter Cheeseman's (1985) work "In Defense of Probability" and Judea Pearl's (1988) "Probabilistic Reasoning in Intelligent System" [5, 6]. A new approach, the Bayesian network formalism, was invented to allow not only efficient representation of uncertain knowledge but also rigorous reasoning with them. The new approach now dominates AI research area on uncertain reasoning and expert systems because it largely overcomes many problems of the systems of the 1960s and 1970s [7].

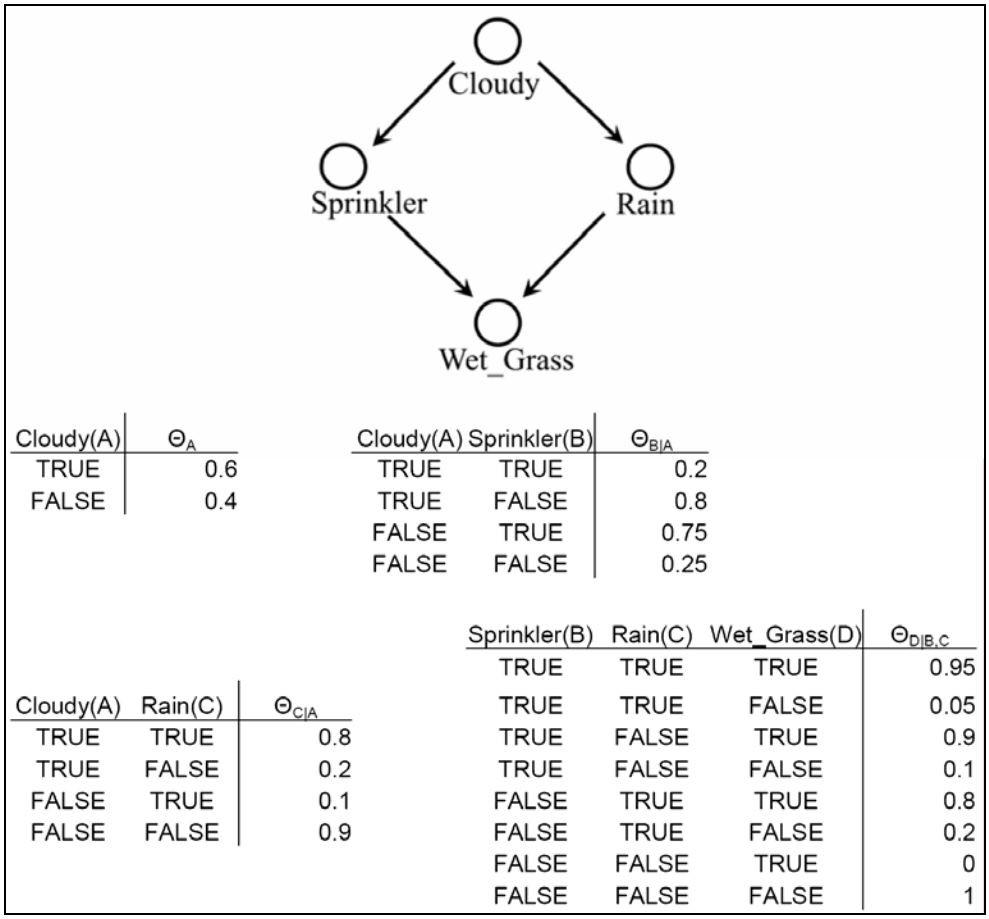
Bayesian networks (also called belief networks sometimes) are powerful tools for modeling causes and effects in a wide variety of domains. By using a graphical model, a Bayesian network is to represent relationships among variables of interests. Briefly, Bayesian network is a model that can be used to model anything, such as the weather, a disease and so on. It is very effective for modeling vague, incomplete and uncertain situations [8, 9].

Graphically, Bayesian networks are models in which each variable is represented by a node, and causal relationships are denoted by an arrow, which is called an edge. The direction of the arrow indicates the direction of causality. The intuitive meaning of an edge drawn from node X to node Y is that node X has a direct influence on node Y. When two nodes are joined by an edge, the causal node is called the parent of the other node. Figure 1 is a sample of Bayesian network. In Figure 1, Cloudy is a parent of Rain, and Rain is the child of Cloudy. Child nodes are conditionally dependent upon their parent nodes. Next, Bayesian network graphic model is explained via the sample in Figure 1 [10].

Variables in this Bayesian network could be whether the sprinkler is on, Cloudy, Rain or whether grass is wet. Each node has different states or a set of probable values. For

example, the weather could be cloudy or sunny, Sprinkler could be on or off, the weather could be raining or not, and grass could be wet or dry. There is some causality in the network: If the weather is rainy, it will make the grass wet directly. However, the grass could also be wet even in sunny weather by a homeowner turning on the sprinkler.

Not only Bayesian network is a model to represent the possible states of a given domain, but also contains probabilistic relationships among some of the states of the domain. In a Bayesian network, the state of some nodes may affect probabilities of other nodes. The effect is dependent on the causal relationships among the nodes. Similarly, the chance that a node is in one state is dependent on the state of another node according to prior information about the relationships among nodes. In order to describe the probability of every node in the Bayesian network, a *conditional probability table* (CPT) is used. A set of CPT $\Theta$ are showed in Figure 1. For every variable X in the graph G, assume Y is X's parents, we need to provide the probability of x given y, Pr(x|y), for each value x of X and each instantiation y of parents Y. We give the CPT of Pr(a), Pr(b|a), Pr(c|a) and Pr(d|bc) in the Figure 1 [11].




| Cloudy(A) | $\Theta_A$ |
|---|---|
| TRUE | 0.6 |
| FALSE | 0.4 |

| Cloudy(A) | Sprinkler(B) | $\Theta_{B\|A}$ |
|---|---|---|
| TRUE | TRUE | 0.2 |
| TRUE | FALSE | 0.8 |
| FALSE | TRUE | 0.75 |
| FALSE | FALSE | 0.25 |

| Cloudy(A) | Rain(C) | $\Theta_{C\|A}$ |
|---|---|---|
| TRUE | TRUE | 0.8 |
| TRUE | FALSE | 0.2 |
| FALSE | TRUE | 0.1 |
| FALSE | FALSE | 0.9 |

| Sprinkler(B) | Rain(C) | Wet_Grass(D) | $\Theta_{D\|B,C}$ |
|---|---|---|---|
| TRUE | TRUE | TRUE | 0.95 |
| TRUE | TRUE | FALSE | 0.05 |
| TRUE | FALSE | TRUE | 0.9 |
| TRUE | FALSE | FALSE | 0.1 |
| FALSE | TRUE | TRUE | 0.8 |
| FALSE | TRUE | FALSE | 0.2 |
| FALSE | FALSE | TRUE | 0 |
| FALSE | FALSE | FALSE | 1 |

Figure 1. A Bayesian network over four variables and its conditional probability table

The following is a definition of Bayesian network.

**Definition (Bayesian network):** A Bayesian network for variables X is then a pair (G, Θ), where,

- G is a directed acyclic graph over variables X;
- Θ is a set of conditional probability tables, one table $\Theta_{X|Y}$ for each variable X and its parents Y. the number assigned by CPT $\Theta_{X|Y}$ to the conditional probability *Pr(x|y)* is denoted by $\theta_{x|u}$. It is required that $\sum_x \theta_{x|y} = 1$ for every parent instantiation y.

The full specifications for Baysian network is as follows [7, 12]:

1. The random variables, either discrete or continuous, correspond to the nodes of the network.
2. There are a set of directed links or arrows between the nodes of the network. If there is an arrow from node A to node B, A is said to be a patent of B, and B is a child of A.
3. The conditional probability distribution of each node $X_i$ is represented as $P(X_i|Parents(X_i))$. It quantifies the effect of the parents on the node.
4. The local probability distribution of node $X_i$ is *unconditional* if it has no parents. Otherwise, it is *conditional* [12].
5. A node is defined as an evidence node if value of the node is observed [12].
6. The graph has no directed cycles (and hence is a directed acyclic graph, or DAG).
7. The joint distribution of the node values can be written as the product of the local distributions of each node and its parents [7].

$$P(X_1,...,X_n) = \prod_{i=1}^{n} P(X_i \mid parents(X_i))$$

Bayesian network also encodes independencies between variables. The graphical property of *d-separation* can determine conditional independence. If two sets of nodes X and Y are d-separated in the graph by a third set Z, then the corresponding variable sets X and Y are independent given the variables in Z. The definition of *d-separation* is given.

**Definition (*d-separation*):** Let X, Y and Z be disjoint sets of nodes in a DAG *G*. We will say that X and Y are *d-separation* by Z, if and only if every path between a node in X and a node in Y is blocked by Z.

The idea of blocking a path by Z is important to understand the definition of *d-separation*, where a path is blocked by Z if at least one value on the path is closed given Z. One way to understand the notion of blocking better is to use a model of pipe (the path) and valve (each variable). A valve on the pipe can be either open or close depending on some conditions. The path is unblocked only when all valves are open and is blocked when one or more valves are closed. Only one closed valve is needed to block the whole path. Once the conditions to close a valve are given, blocking of a path is determined.

## 1.2 Causal Bayesian Networks

A Bayesian network uses directed acyclic graph (DAG) to represent relationships of variables. In a DAG, each node represents a variable and each arc represents probabilistic influence. We can gather information about probabilities of events and changes of probabilities with subsequent observations from a joint distribution. However, a probabilistic system can not predict what will happen if there are environmental changes or external interventions, because such predictions can not be obtained from probabilistic information alone, no matter how specific the information is. Predictions are from the causal understanding of the underlying processes. Because a Bayesian network is a carrier of the conditional independencies of a set of variables, it does not necessarily represent causal connections among variables. But the related causal Bayesian network (or *causal network* for short) can be used to model causal relations closely [12]. A causal Bayesian network is a

Bayesian network in which the arrow between a parent node and a child node represents a direct causal influence relative to other nodes in the network [13]. In a causal model, once we know the identity of the mechanism altered by an intervention and the nature of the alteration, the overall effect of an intervention can be predicted by modifying the corresponding factors and using the modified product to compute a new probability function [14].

The following is an intervention example:

To represent the action "turning the sprinkler On". We delete the link $X_1 \rightarrow X_2$ from Figure 1 and assign $X_2$ the value "On".

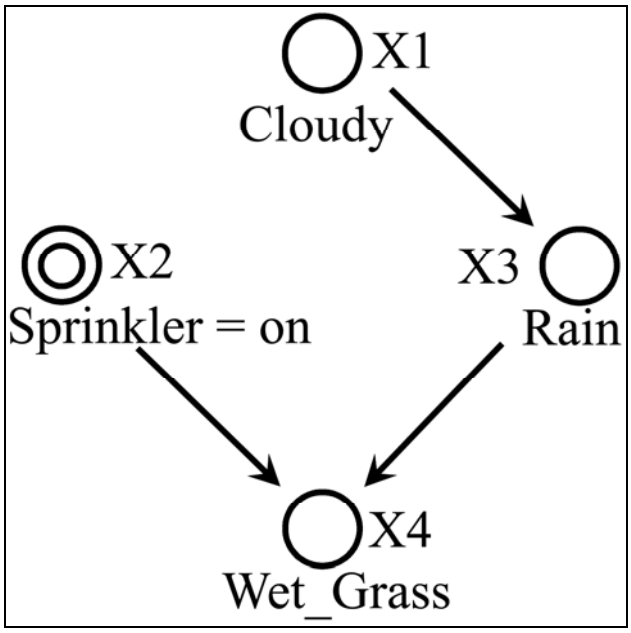



Figure 2. Intervention example graph.

Now, the joint distribution of the node values can be written as the following product:

$$P_{X_2=On}(x_1, x_3, x_4) = P(x_1)P(x_3 \mid x_1)P(x_4 \mid x_3, x_2 = On)$$

The deletion of the factor $P(x_2|x_1)$ represents that whatever relationship existed between Cloudy and Sprinklers prior to the action, that relationship is no longer in effect after the action was performed. Once we physically turn the sprinkler on and keep it on, a new mechanism determines the state of the sprinkler. We should note that the action $do(X_2 = On)$ and the observation $X_2$ = On are different. According to the definition of atomic intervention: $do(X_2 = On)$ means to fix Variable $X_2$ to be *On*.

**Definition (Atomic intervention do (T = t)):** fixing a set T of variables to some constants T = t.

We can use the causal Markov assumption to connect causation with probability distributions. In the causal Markov assumption, the probability distribution P generated by a causal diagram G satisfies the causal Markov condition: each variable $V_i$ is independent on all its non-descendants, given its patents $Pa_i$ in G

$$P(x_1,...,x_n) = \prod_i P(x_i \mid pa_i)$$

We can say that G and P are compatible or P is Markov relative to G, if above factorization is true for a probability distribution function P in a DAG G [15, 16].

From the description above, a definition of Causal Bayesian Network is given.

**Definition (Causal Bayesian Networks):** let $P(x)$ be a probability distribution on a set $X$ of variables, and let $P_y(x)$ denote the distribution resulting from the intervention $do(Y = y)$ that sets a subset $Y$ of variables to constants $y$. Denote $\boldsymbol{P}_*$ as the set of all interventional distributions $P_y(x)$, $Y \subseteq X$, including $P(x)$, which represents no intervention (*i.e.*, $Y = \Phi$). A DAG $G$ is said to be a causal Bayesian network compatible with $\boldsymbol{P}_*$ if and only if the following three conditions hold for every $P_y \in \boldsymbol{P}_*$:

i) $P_y(x)$ is Markov relative to G ( $P_x(v) = \prod_i P_x(v_i \mid pa_i)$ );

ii) $P_y(x_i) = 1$ for all $X_i \in Y$ whenever $x_i$ is consistent with $Y = y$;

iii) $P_y(x_i \mid pa_i) = P(x_i \mid pa_i)$ for all $X_i \notin Y$ whenever $pa_i$ is consistent with $Y = y$.

In this thesis, we focus on introducing some algorithms of causal reasoning in Causal Bayesian Networks and our software system. In the next chapter, we present the definitions and notations and lemmas used to support identify algorithms in this thesis. In chapter 3, three systematic procedures (identify causal effect of T on S, conditional causal effects of T on S conditioned on another set C and finding functional constraints) , which are implemented in our software system, are introduced. In chapter 4, we present development and implementation of our software system. In chapter 5, we present testing analysis and result of our software system.

# CHAPTER 2. DEFINITIONS

In this thesis, upper letters, such as V, are used for variable sets; lower letters, such as v, are used for the instances of variable set V; upper letters, like X, Y and $V_{i,}$ are used for single variable; x, y and $v_i$ can be presented as their value. In this chapter, definitions used in this thesis are interpreted.

## 2.1 Introduction

One of the most common causal models for encoding distributional and causal relationships is causal Bayesian Network (also known as Markovian model). A Markovian model consists of a directed acyclic graph (DAG) G over a set $N = \{N_1, \ldots, N_n\}$ of variables and a set of directed edges. It is called a causal graph. Such a graph model consists two parts of interpretation: probabilistic and causal interpretation. Markovian model is also called causal Bayesian network, because a Bayesian network is just a Markovian model that holds the probabilistic interpretation [17].

There are four elements in a Markovian model:

$$M = < V, U, G_{V \cup U}, P(v_i \mid pa(V_i)) >$$

Where V is a set of observed variables; U is a set of unobserved variables; G is a DAG graph, which includes variables $V \cup U = N$; P is the conditional probability of variable V given its parents P.

The probabilistic interpretation says that each variable is independent of all its non-descendants given its direct parents in the graph. Based on the probabilistic interpretation, the

joint probability function P(v) = P ($v_1$, …, $v_n$) can be factorized as product in equation (1) [18]:

$$P(v) = \prod_{V_i \in V} P(v_i \mid pa(V_i)) \tag{1}$$

The causal interpretation says that the directed edges in G represent causal influences between the corresponding variables. This assumption enables us to predict the intervention effects. Interventions are specific modifications of some factors in the above equation (1). For example, fixing a set T $\subseteq$ V of variables to some constants T = t, denoted by do(T = t) or do(t), is the simplest kind of intervention. The post-intervention distribution yielded by the intervention is equation (2):

$$P_t(v) = \begin{cases} \prod_{V_i \in V \setminus T} P(v_i \mid pa(V_i)) & v \ \ consistent\ with \ \ t \\ 0 & v \ \ inconsistent \ \ with \ \ t \end{cases} \tag{2}$$

Where $V_i \in V \setminus T$ means $V_i \in V$ *and* $V_i \notin T$.

If let T be a set of treatment variables and Y is V\T, we can calculate the probability $P_t(y)$ from equation (2). The quantity is called the causal effect of T on Y. If all the variables V were observed, all causal effects in a given causal graph G would be computable.

V and U represent the sets of observed and unobserved variables in graph G respectively. The observed probability distribution P(v) is given in equation (3):

$$P(v) = \sum_{U} \prod_{V_i \in V} P(v_i \mid pa(V_i)) \prod_{V_j \in U} P(v_j \mid pa(V_j)) \tag{3}$$

The post-intervention distribution Pt(v) is a mixture of truncated product given in equation (4):.

$$P_t(v) = \begin{cases} \sum_U \prod_{V_i \in V \setminus T} P(v_i \mid pa(V_i)) \times \prod_{V_j \in U} P(v_j \mid pa(V_j)) & v \ \ consistent \ \ with \ \ t \\ 0 & v \ \ inconsistent \ \ with \ \ t \end{cases} \quad (4)$$

If we only want to know the post-intervention distribution for an observed variable subset S $\subset$ V, Pt(S) is given by equation (5):

$$P_t(s) = \begin{cases} \sum_{V_j \in (V \setminus S) \setminus T} \sum_U \prod_{V_i \in V \setminus T} P(v_i \mid pa(V_i)) \times \prod_{V_j \in U} P(v_j \mid pa(V_j)) & v \ \ consistent \ \ with \ \ t \\ 0 & v \ \ inconsistent \ \ with \ \ t \end{cases} \quad (5)$$

We define Q[S] as a causal effect of a set of observed variables T on a set of observed variables S, here T is V\S and V is all of observed variables in causal graph G [19]. In other words, the quantity Q[S] denotes the post-intervention distribution of S under an intervention of all other variables.

$$\begin{aligned} Q[S] &= P_{V \setminus S}(S) = P_t(S) \\ &= \sum_{u(S)} \prod_{\{i \mid V_i \in S\}} P(v_i \mid pa_{v_i}) \prod_{\{i \mid U_i \in U(S)\}} P(u_i \mid pa_{u_i}) \end{aligned} \quad (6)$$

Q[S] is a function of S, the observed parents of S, and the observed parents of U(S). U(S) is a set of hidden variables that are ancestors of S in graph $G_{S \cup U}$ [17, 19].

## 2.2 Definitions

In this section, we will introduce some definitions that are important in this thesis.

**Definition 1 (Causal –Effect Identifiability $P_t(s)$ )**

The causal effect of a set of variables T on a disjoint set of variables S is said to be identifiable from a graph G if all the quantities $P_t(s)$ can be computed uniquely from any positive probability of the observed variables-- that is, if $P_t^{M_1}(s) = P_t^{M_2}(s)$ for every pair of models M1 and M2 with $P_t^{M_1}(v) = P_t^{M_2}(v) > 0$ and G ($M_1$) = G (M2) = G [17].

**Definition 2 (Conditional Causal –Effect Identifiability $P_t(s \mid c)$ )**

The causal effect of a set of variables T on a disjoint set of variables S conditioned on another set C is said to be identifiable from a graph G if all the quantities $P_t(s \mid c)$ can be computed uniquely from any positive probability of the observed variables-- that is, if $P_t^{M_1}(s \mid c) = P_t^{M_2}(s \mid c)$ for every pair of models M1 and M2 with $P_t^{M_1}(v) = P_t^{M_2}(v) > 0$ and G ($M_1$) = G (M2) = G [20].

Generally, Pa($V_i$) is used to denote parent nodes set of variable $V_i$ in a directed graph G and pa($V_i$), or $pa_i$ is used to denote an instance of Pa($V_i$); Ch($V_i$) is used to denote children nodes set of variable $V_i$ in a directed graph G and ch($V_i$) is used to denote an instance of Ch($V_i$).

We let An(C) denote the union of C and the set of ancestors of the variables in C, for any set of C. $An^u(C)$ denote the set of hidden variables in An(C), $An^u(C) = An(C) \cap U$. $An^v(C)$ denote the set of observed variables in An(C), $An^v(C) = An(C) \cap V$. So we have $U(S) = An^u(S)_{G_{S \cup U}}$ from the definition of U(S).

If a set $A \subseteq V$ contains its own observed ancestors, $A = An^v(A)$ , A is called an *ancestral set*. Similarly, if a set $A \subseteq V$ contains its own observed descendants, $A = De^v(A)$ , A is called a *descendent set.*

**Definition 3 (c-component relation)**

c-component (confounded component) relation is defined on the unobserved variable set U of graph G as:

For any unobserved variable $U_1$ and $U_2$, they are related under the c-component relation if and only if at least one of conditions below is satisfied:

i). There is an edge between $U_1$ and $U_2$.

ii). Both $U_1$ and $U_2$ are parents of the same observed variable.

iii). Both $U_1$ and $U_2$ are in the c-component relation with respect of another unobserved variable $U_3$.

From the definition, we know that c-component relation is reflexive, symmetric and transitive.

A c-component of variable set S on graph G includes all the unobserved variables that are under the same c-component relation. It also includes all the observed variables that have an unobserved parent, which must be a member of that c-component. Obviously, any observed variable belongs to only one c-component. If an observed variable does not have an unobserved parent, the observed variable is the only variable appearing in the c-component [17].

# CHAPTER 3. ALGORITHMS

In this chapter, we will introduce three identifying algorithms we used in the software program.

## 3.1 Identification of Causal-Effect in Causal BNs

In this section, a function of this software program, which is to identify causal effects from nonexperimental data in a causal Bayesian Network, will be introduced.

The identifiability function accesses strength of causal-effect relationships from a causal Bayesian Network and gives the total causal effect in terms of estimable quantities. It is to compute the probability of a set of variables given intervention on another set of variables in a causal graph. In other words, the software program has been devised to show whether a causal-effect, denoted as Pt(S), is identifiable or not. If the Pt(S) is identifiable, the software program will calculate the causal-effect value. The identifiability method used in this software is created by Tian and Pearl [19], where both algebraic and graphic methods are used. In Tian and Pearl's algorithm, a causal graph is a semi-Markovian graph, where every unobserved variable (hidden) is a root and has exactly two observed variable children [21]. The algorithm is sound and complete even when the algorithm is used in a general causal graph, which has been proved by Huang and Valtorta. A similar result was provided by Shpitser and Pearl [17, 22].

In the software program, a causal graph G should be created before a user deploys the function. V stands for all the observed variables in G. Then the user need to set both a set of effect variables (denoted as S) and a set of intervention variables (denoted as T). In the

meantime, the user should know whether a given variable is observed or not in the created causal Bayesian network. In our causal graph, S and T are disjoint observed variable sets.

The identifiability algorithms are presented in Figure 3, Figure 4 and Figure 5 based on the lemmas in the Appendix A. Those algorithms are sound and complete. They have been implemented in the software program.

**Algorithm Computing Pt(S)**

INPUT: two disjoint observable variable sets S, T $\subset$ V.

OUTPUT: the expression for Pt(S) or Fail

1. Let D = An(S) $_{GV\backslash T}$ $\cap$ V
2. Using the Computing Q[S] algorithm shown in Figure 4 to compute Q[D]
3. If the algorithm returns FAIL, then output FAIL.
4. Else, output Pt(S) = $\sum_{D\backslash S} Q[D]$

Figure 3. Algorithm Computing Pt(S)

**Algorithm Computing Q[S]**

INPUT: S $\subseteq$ V

OUTPUT: Expression for Q[S] or FAIL.

Let V be partitioned into $V_1$, …, $V_k$, each of them belonging to a c-components in G, and S be partitioned into $S_1$, …, $S_l$, each of them belonging to a c-components in Gs, and $S_j \subseteq V_j$, We can:

i), Compute each $Q[V_j]$ with lemma 2.

ii), Compute each $Q[S_j]$ with Identify algorithm above with

C = $S_j$, T = $V_j$, $Q = Q[V_j]$.

iii), If in ii), we get FAIL as return value of Identify algorithm of any $S_j$, then Q[S] is unidentifiable in graph G; else $Q$[S]is identifiable and $Q$[S] = $\prod_{j=1}^{l} Q[S_j]$.

Figure 4. Algorithm Computing Q[S]

**Algorithm Identify(C, T, $Q$)**

INPUT: C $\subseteq$ T $\subseteq$ V, $Q = Q[T]$,

$G_T$ and $G_C$ are both composed of one single c-component.

OUTPUT: Exptession for $Q[C]$ in terms of $Q$ or FAIL.

Let A= An(C) $_{GT} \cap$ T

i), If A = C, output $Q[C] = \sum_{T \setminus C} Q[T]$ (Cf. lemma 1)

ii), If A = T, output FAIL.

iii), If C $\subset$ A $\subset$ T

1. Assume that in $G_A$, C is contained in a c-component $T_1'$, $T_1 = T_1' \cap A$
2. Compute $Q[T_1]$ from $Q[A] = \sum_{T \setminus A} Q[T]$ (Cf. lemma 2)
3. Output Identify(C, $T_1$, $Q[T_1]$)

Figure 5. Algorithm Identify(C, T, $Q$)

In figure 5, case i) and case iii) always work. Case ii) is determined by the following lemma 4 [17].

Lemma 4: In a general Markovian model G, if

1. G itself is a c-componen
2. $S \subset V$ and $G_S$ has only one c-component
3. All variables in V\S are ancestors of S

Then Q[S] is unidentifiable in G.

## 3.2 Identifying Conditional Causal-Effect

In this section, a software function identifying conditional causal effect is introduced. This problem is important for evaluating conditional plans and stochastic plans [23]. The function is to assess causal effect relationships from non-experimental data and theoretical assumptions. Those assumptions are encoded in the form of a DAG including unobserved variables. A systematic procedure identifying causal effects between two sets of variables conditioned on another set of variables was provided by Tian. The procedure relies heavily on results computed by the algorithm in section 3.1, which only concerns with identifying unconditional causal effects [20]. The identifiable conditional causal effects are expressed by factorization of the observed joint distribution [20]. The algorithm, identifiable conditional causal effects, has been proven to be sound and complete and is implemented in our software system.

In the software program, a causal Bayesian Network G should be created or loaded before a user deploys the function. V stands for all the observed variables in G. Then the user need to set both a set of effect variables (denoted as S) and a set of intervention variables (denoted as T) and a set of conditional variables (denoted as C). In the meantime, the user should know that every variable is observed or unobserved. In our causal graph, S, T and C are disjoint observed variable sets.

We have equation 13 from *Bayes conditioning*.

$$\mathrm{P}_t(s \mid c) = \frac{P_t(s,c)}{P_t(c)} \tag{13}$$

Therefore,

1). If Pt(s,c) is identifiable, so is Pt(s|c). Then we can calculate Pt(s,c) with the procedure provided by Tian [21]. We have introduced the procedure in detail in the section 3.1.

2). If Pt(s,c) is not identifiable but Pt(c) is, then Pt(s|c) is not identifiable. However, if there is no variable in C belongs to T's descendant variables set, that is, Pt(c) = P(c), and then Pt(s|c) is identifiable if and only if Pt(s,c) is identifiable.

3). When both Pt(s,c) and Pt(c) are not identifiable, Pt(s|c) could be either identifiable or unidentifiable. That will be depended on whether the unidentifiable terms can be canceled out in the expression for Pt(s,c) and Pt(c).

In order to assess Pt(s|c), Pt(s,c) and Pt(c) should be computed first. We use the method described in section 3.1. Then we have

$$P_t(s,c) = \sum_f \prod_i Q[D_i] \tag{14}$$

$$P_t(c) = \sum_s P_t(s,c) = \sum_s \sum_f \prod_i Q[D_i] \tag{15}$$

Where, $F = D \setminus (S \cup C)$ and $D = An(S \cup C)_{G_{V \setminus T}}$ ; $S_1,...,S_k$ are the c-components partitioned in V; $D_1,...,D_l$ are the c-components partitioned in subgraph $G_D$; Each $D_i$ is a subset of $S_j$ and $Q[D_i]$ can be calculated by algorithm $Identify(D_i, S_j, Q[S_j])$, which is shown in figure 5. $Q[S_j]$ can be calculated by Lemma 3 in Appendix A.

Let I stand by the sets of $D_i$'s that are identifiable, and let N stand by the sets of $D_i$'s that are unidentifiable. We decompose F into $F_0$ and $F_1$, I into $I_0$ and $I_1$. $F_0$ is over unidentifiable terms and $F_1$ over indefinable terms. Therefore, we can rewrite the equation (14) and (15) as:

$$P_t(s,c) = (\sum_{f_0} \prod_{D_i \in N} Q[D_i] \prod_{D_i \in I_0} Q[D_i])(\sum_{f_1} \prod_{D_i \in I_1} Q[D_i]) \tag{16}$$

$$P_t(c) = (\sum_{s} \sum_{f_0} \prod_{D_i \in N} Q[D_i] \prod_{D_i \in I_0} Q[D_i])(\sum_{s} \sum_{f_1} \prod_{D_i \in I_1} Q[D_i]) \tag{17}$$

We like $F_0$ and $I_0$ including as fewer elements as possible. The method we used to partition F and I is shown in Phase 3 of the Figure 6 (Algorithm computing conditional causal effects, Pt(s|c)).

If the unidentifiable terms $\sum_{f_0} \prod_{D_i \in N} Q[D_i] \prod_{D_i \in I_0} Q[D_i]$ could be totally canceled out by $\sum_{s} \sum_{f_0} \prod_{D_i \in N} Q[D_i] \prod_{D_i \in I_0} Q[D_i]$, we can get $P_t(s \mid c)$ is identifiable and it can be represented as equation (18). Otherwise, $P_t(s \mid c)$ is not identifiable.

$$\mathrm{P}_t(s \mid c) = \frac{P_t(s,c)}{P_t(c)} = \frac{\sum_{f_1} \prod_{D_i \in I_1} Q[D_i]}{\sum_{s} \sum_{f_1} \prod_{D_i \in I_1} Q[D_i]} \tag{18}$$

The procedure of identifying conditional causal effects is explained above. The following is the algorithm of computing Pt(s|c). There are 4 Phases in the procedure. In Phase-1, we define variable sets D and F in the graph G; the causal effect of each set of c-component variables of G, $Q[S_i]$, is computed; each c-component set $D_i$ of subgraph $G_D$ are found. In Phase-2, we find the c-component set $S_j$ of G, which is compatible with $D_i$; each $Q[D_i]$ is calculated. If $Q[D_i]$ is identifiable then put $D_i$ into set I. Otherwise put $D_i$ into N. In Phase-3, *F* is partitioned into *F1* and *F0*; *I* is partitioned into *I0* and *I1*; at the meantime, *F0* and *I0* are tried to contain as fewer elements as possible. In Phase-4, to determine whether the unidentifiable term can be cancelled out from Pt(s,c) and P(c) totally, and then determine Pt(s|c) is identifiable or not [20].

**Algorithm 1(Computing Pt(s|c))**

INPUT: Three disjoint sets $T \subset V$, $S \subset V$ and $C \subset V$.

OUTPUT: The expression for Pt(s|c) or FAIL.

Phase-1:

1. Find the c-components of G: $S_1,\ldots,S_k$.
2. Compute $Q[S_1],\ldots,Q[S_k]$ by Lemma 3.
3. Let D = $An(S \cup C)_{G_{V \setminus T}}$ and $F = D \setminus (S \cup C)$.
4. Find the c-components of the sub graph $G_D$: $D_1,\ldots,D_l$

Phase-2:

1. For each set $D_i$:

   Find the set $S_j$ to which $D_i$ belongs. Call algorithm Identify($D_i$, $S_j$, $Q[S_j]$).

   If the algorithm returns FAIL, then put $D_i$ into the set I.
2. If N is empty , then stop and output

$$P_t(s \mid c) = \frac{\sum_f \prod_i Q[D_i]}{\sum_s \sum_f \prod_i Q[D_i]}$$

Phase-3:

1. Initialize $F_0 = F \cap (U_{D_i \in N} Pa(D_i))$, $F_1 = F \setminus F_0$, and $I_0 = \phi, I_1 = I$.
2. For each $D_i \in I_1$:

   If $Pa(D_i) \cap F_0 \neq \phi$, then remove $D_i$ from I1 and put it into I0.
3. Let $B = F_1 \cap U_{D_i \in I_0} Pa(D_i)$
   - If B is not empty, remove variables in b from $F_1$ and put them into $F_0$. Then go back to step 2.
   - If B is empty, then continue to Phase-4.

Phase-4:

If $B = F_1 \cap U_{D_i \in I_0} Pa(D_i)$, then

Output the expression for Pt(s|c) as given in Eq **(;;)**

Else

Output FAIL

Figure 6. An algorithm for identifying conditional causal effects, Pt(s|c)

## 3.3 Identifying Constraints Implied by Causal Models

Causal model may impose two types of constraints, namely conditional independencies and functional constraints. Conditional independencies can be read via d-separation criterions, but functional constraints are not available via general graphic criterions. In this section, a systematic method of finding such functional constraints is introduced. The systematic procedure was developed by Tian and Pearl, and is implemented in our software system [19]. The algorithm has been proven to be sound.

Identifying non-independence constraints is very helpful for validating causal models and for noticing or understanding the differences between causal models, where they have the same set of conditional independence relationships among the observed variables [19]. For example, the two models in Figure 7 are the same set of independence statements (A is independent of C given B) but their Verma's constraint is different. In figure 7(a), we can know that $\sum_b P(d \mid a,b,c)P(b \mid a)$ is not a function of $\boldsymbol{a}$ by a simple analysis, because there is no relation between node A and node D after eliminate node B from Figure 7 (a). But it is a function of $\boldsymbol{a}$ obviously in Figure7 (b) [24].

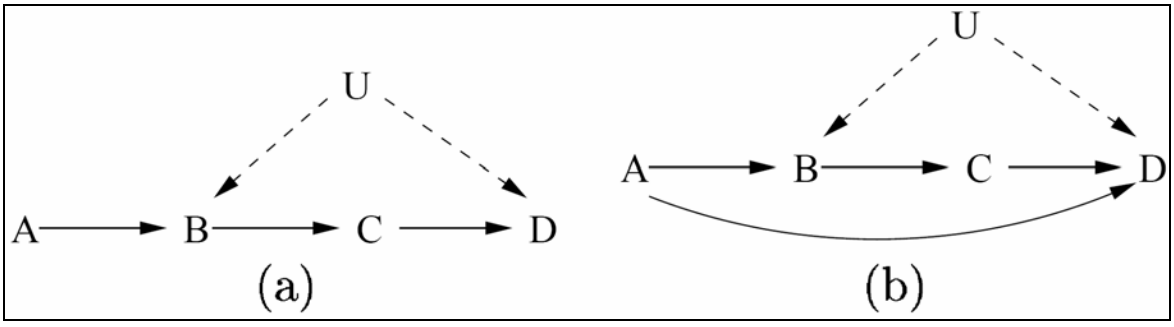



Figure 7. Identifying non-independence constraints

The following is the procedure of identifying constraints systematically. Lemma 1, Lemma 2 and Lemma 3 (introduced in Appendix A) are applied in this procedure.

Let $V^{(i)} = \{V_1,...,V_i\}$, i = 1,…,n. where $V_1 < ... < V_i$ they are in topological order.

For i = 1 to n

We look for constraints that involve $V_i$ and the variables before $V_i$ in the topological order.

(A1) If the subgraph $G(V^{(i)})$ has more than one c-component, assuming that $V_i$ is in the c-component $S_i$ of $G(V^{(i)})$, $Q[S_i]$ may be calculated from P(v) by Lemma 3 and give a conditional independence constraint. $V_i$ is independent of its predecessors given its effective parents, variables in $S_i$ other than $V_i$, and the effective parents of other variables in $S_i$. That is, $V_i$ is independent of $V^{(i)} \setminus Pa^+(S_i)$ given $Pa^+(S_i) \setminus \{V_i\}$.

(A2) Consider $Q[S_i]$ in the subgraph $G(S_i)$

For each descendent set $D \subset S_i$ in $G(S_i)$, $V_i$ is not in D. We have the following equation by Lemma 1.

$$\sum_d Q[S_i] = Q[S_i \setminus D]$$

If some effective parents of D are not effective parents of $S_i \setminus D$, then a constraint on the distribution will be implied, that is,

$\sum_d Q[S_i]$ is independent of $(Pa^+(S_i) \setminus D) \setminus Pa^+(S_i \setminus D)$

Let $D' = S_i \setminus D$.

Consider $Q[D']$ in the subgraph $G(D')$

If $G(D')$ has more than one c-component, assuming that $V_i$ is in the c-component $E_i$ of $G(D')$, $Q[E_i]$ may be calculated from $Q[D']$ by Lemma 2. It is understandable that $Q[D']/\sum_{v_i} Q[D']$ is a function only of $Pa^+(E_i)$.

If $Pa^+(D') \setminus Pa^+(E_i) \neq \phi$ then a constraint on the distribution P(v) is imposed.

Repeat the process (A2) on $Q[E_i]$, we use $E_i$ substitute for $S_i$.

In the above procedure, effective parent is an observed variable's parent or there is a directed path from it to the observed variable. For example, if an observed variable $V_i$ is a parent of $V_j$ or there is a directed path from $V_i$ to $V_j$, we call $V_i$ is an effective parent of $V_j$. $Pa^+(S)$ denotes the union of S and the set of effective parents of S for any set $S \subseteq V$.

# CHAPTER 4. SOFTWARE DEVELOPMENT AND IMPLEMENTATION

The requirements of our software system, design methods and implementation processes are introduced in this chapter.

## 4.1 Requirement

In this section, we list and rationalize the requirements for our BNs identification software. The requirements are organized into functional and non-functional groups. Functional requirements are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. Non-functional requirements, as the name suggests, are those requirements which are not directly concerned with specific functions delivered by the system. Non-functional requirements are constraints on the services or functions offered by the system [25].

### 4.1.1 Functional Requirements

Functional requirements are more concrete than their non-functional requirements. Those requirements for a system describe the functionality or services that the system is expected to provide.

- Since our software provides analysis of causal effect in BNs for Bayesian Network researchers, users are able to create and edit Causal graphs and read graphic models from an appropriate window.
- The software system can save graphic models to a XML file following a *Bayesian Network Interchange Format.*

- The software system provides appropriate viewer for users to read the solutions of software analysis.
- The software system provides friendly interface for users to operate the functions of software.
- The software system provides error handler. Once users use a software function before complete all operations for satisfying the function, error messages will be printed.

### 4.1.2 Non-functional Requirements

- This software system contains help tools for Bayesian Network researchers. It can run on multi- platforms, such as Windows, Linux and Unix OS. It also can be adopted by browsers.
- To save the software development cost and development time, we use free resources from website. The software program makes use of the free resources as much as possible.
- The algorithms implemented in our software have been proven to be sound and completed or sound.
- Running time of the algorithms implemented in our software is polynomial.
- Space used in the algorithms implemented in our software is polynomial.
- Graphic models are saved in XML files. They consist with some standard interchange formats of Bayesian Networks, such as the Bayesian Interchange Format version 0.1 (BIF 0.1), BIF 0.15 and XMLBIF 0.3. The software can open any graphic model in those standard formats.

- The software system is easy to use and maintenance.

## 4.2 Design

Software design is a process of problem-solving and planning for a software solution. After the requirements of software are determined, we design architecture view for our software system. In this section, the design of our software is introduced.

### 4.2.1 System Model

The architectural design process is to establish a basic structural framework for a system. As a part of the system requirements and design activity, it is necessary for the architectural design process to model a set of components and relationships among these components. The system architecture is usually presented as a block diagram showing the major components and the interconnections between them. Each component is represented by a rectangle and the relationship between those components is indicated by arrows linking those rectangles [25]. Figure 8 shows the decomposition of our system into its principal components.

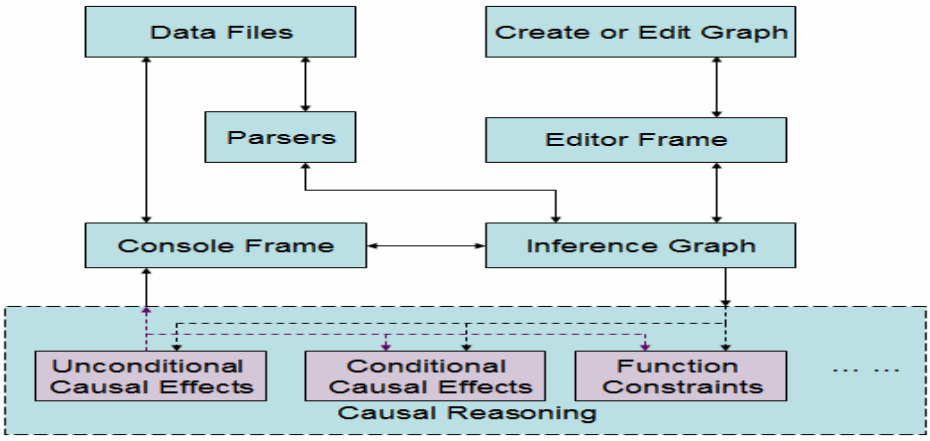



Figure 8. The system architecture of the software system

The Editor Frame and Console Frame are two user interfaces. Editor Frame obtains graphic information from Data files via Parsers and Inference Graph component. Conversely, it sends graphic information to Data files through the two components too. Create or Edit Graph component can edit causal graph by clicking the buttons on Editor Frame. Console Frame controls the software system via Inference Graph, since it holds all current causal BN information. Console Frame communicates with Data File when the system needs save or load a data file. Table 1 is the brief function descriptions of each component.

The system architecture is compact and manageable. It describes how the system is organized and how the components interoperation [25]. The design satisfies the function requirements of our software system.

**Table 1**. Function Description of components

| **Component Name** | **Component Description** |
|---|---|
| Data Files | Xml Files with standard BN Interchange Formats |
| Create or Edit Graph | A set of tools for Creating and Editing Graphic Models, such as create node, delete node, create arc, move node, and so on. |
| Parsers | Parsers hold definitions of the Interchange Format. It let system obtain the parsed Bayesian networks. |
| Editor Frame | It is an interface. The interface contains some graphic model edit buttons. The causal BN graphs are displayed in the frame. |
| Console Frame | It is an interface. The interface contains some menus, which are used to operate a variety of functions of our software system. Error messages, instruction messages and analysis results are shown in the frame. |
| Inference Graph | Inference Graph component contains all the information about the graphical structure of a causal Bayesian network. It can control system operations by changing operation models, such as identify H_mode (Pt(s) mode) or create mode, and so on. |
| Causal Reasoning | Causal Reasoning contains the components, which can solve causal reasoning problems, such as identify causal effects in causal Bayesian networks. |

### 4.2.2 Design Considerations

. The software design should reflect the requirements of the software. The followings are some aspects we considered during our design:

- Reliability - The software is able to perform required functions under stated conditions for a specified period of time. All of the function algorithms implemented in this system are proved be correct.
- Extensibility - New capabilities can be added to the software without major changes to the underlying architecture.
- Robustness - The software is able to operate under stress conditions. It is also be able to tolerate unpredictable or invalid input.
- Fault-tolerance - The software is resistant to and able to recover from component failure.
- Compatibility - The software uses some free resources from internet, such as some classes from software *JavaBays*. We design our system so that it has interoperability with those software resources.
- Modularity - the resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.
- Reusability - the designed modular components should capture the essence of the functionality. This single-minded purpose renders the components reusable wherever there are similar needs in other designs. For example, the implementation for those

lemmas and finding c-component set method are used frequently during programming in our software.

- Portability – users can run the system on any computer that has a java interpreter easily, once they obtain the software system. Users can use the system easily by following the software instruction.

## 4.3 Implementation

Software implementation is to integrate software based components into the organizational structure systematically and effectively. In this section, we introduce the system development environment, data structure and implementation process of our software.

### 4.3.1 Developing Environment

The software system is a full implementation in Java. A Java implementation has several advantages. First, Java was designed to be easy to use so that it is easy to write, compile and debug. Second, a Java package can be exported or run in a variety of platforms such as UNIX, Macintosh and Windows with few modifications. Third, a program or package written in Java can be intimately coupled with World Wide Web pages because Java has been adopted by browsers in the Internet. This feature makes lots of people can access a Java program or package. Forth, a Java package is an excellent tool for people who are interested in using reasoning in network-based applications. Fifth, on Microsoft Windows systems, the Java 2 Runtime Environment's installation program registers a default association for Jar files so that double-clicking a Jar file on the desktop will automatically

run it with `javaw -jar`. Dependent extensions bundled with the application will also be loaded automatically. This feature makes the end-user runtime environment easier to be used on Microsoft Windows systems. Finally, Java is a good object-oriented language. Widgets in java allow researchers to quickly prototype interfaces, and the multi-threaded processing in Java facilities the future parallelization of inference algorithms [2].

The Java version used during the developing is JDK 1.5.0_06, which is available at http://www.eclipse.org/downloads. Eclipse 3.3 is used as our main IDE. It is available at http://www.eclipse.org/downlods. The OS used during development is Microsoft XP Professional, SP2.

Our software interfaces use the main interfaces of JavaBayes, a BN software tools developed by Fabio Gagliardi Cozman, Carnegie Mellon University. We use Abstract Windows Toolkit (AWT) as Java GUI tool kit for compatibility with JavaBayes better. AWT is very stable and standards with every version of Java technology, including Java implementations in old Web browsers. Another advantage is that there is no need to install AWT. It is available with all features everywhere under a Java runtime environment [26].

### 4.3.2 Data Transfer

Data can be locally loaded or saved when the user uses the software as an application. The graph data files use XML-formatted text. Three Bayesian Interchange Formats (BIF 0.1, BIF 0.15 and XMLBIF 0.3) are supported by our software. Figure 9 is a causal BN graph created by our software and Figure 10 is the data file of the graph in XMLBIF 0.3 format.

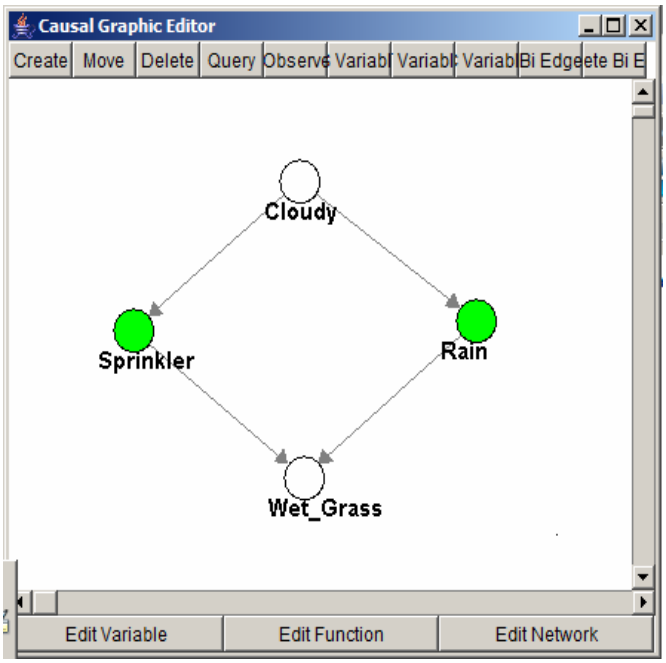



Figure 9. A causal Baysian Network graph created by the software.

```
// Bayesian network
network "Cancer" { //4 variables and 4 probability distributions
}
variable  "Cloudy" { //2 values
        type discrete[2] {  "Present"  "Absent" };
        property "position = (242, 136)" ;
}
variable  "Sprinkler" { //2 values
        type discrete[2] {  "Increased"  "NotIncreased" };
        property "observed Increased" ;
        property "position = (126, 221)" ;
}
variable  "Rain" { //2 values
        type discrete[2] {  "Present"  "Absent" };
        property "observed Present" ;
        property "position = (368, 216)" ;
}
variable  "Wet_Grass" { //2 values
        type discrete[2] {  "Present"  "Absent" };
        property "position = (244, 313)" ;
}
probability (  "Cloudy" ) {
}
probability (  "Sprinkler"  "Cloudy" ) {
}
probability (  "Rain"  "Cloudy" ) {
}
probability (  "Wet_Grass"  "Rain"  "Sprinkler" ) {
}
```

Figure 10. XML Data File for Figure 2 (XMLBIF 0.3 format)

In order to display relevant data, our System software needs parse out the desired parts from the XML files. JavaBayes contains sets of XML parsers. We modified the parsers in JavaBayes, so as to they can save or load the properties of a variable, such as whether the variable is observed or not.

### 4.3.3 Data Structure

We designed our software in such a way that we can add more function components to the software in the future. To satisfy the requirement, we defined different data structure for different function components. We won't have to modify the existed data structure when adding a new function to the software is needed, even the function may require extra features to BNs. By doing this, we can save a lot of coding work when we add new capabilities to the software, because changing existed basic data structure requires lots of code changing.

We use class *BayesNet*, which is from JavaBayes, as Basic data structure to hold all the variables and relationships among them in Bayesian networks. Below is the *BayesNet* class. In the class, *probability_variables* gives the entire features of each variable node. *probability_functions* gives each variable's parent variables.

```
public class BayesNet {
  protected String name;
  protected Vector<String> properties;
  protected ProbabilityVariable probability_variables[];
  protected ProbabilityFunction probability_functions[];
  public final static int INVALID_INDEX = -1;
  public final static int S_INDEX = -1;
  public final static int T_INDEX = -1;
  public final static int C_INDEX = -1;
```

```
    public final static int BIF = 1;
    public final static int XML = 2;
    public final static int BUGS = 3;
}
```

All BN data structures in our software inherit from the basic class *BayesNet*. Class *QuasiBayesNet* is defined in *JavaBayes* by Fabio Gagliardi Cozman. We modified it so that it can recognize hidden variables. We defined class *STBayesNet*, which inherits from class *QuasiBayesNet*, for the $P_t(s)$ function. *STBayesNet* can let function algorithm get more information than *BayesNet*, such as a set of S variables and a set of T variables. Below is the class *STBayesNet*. In the class, we can get the size of S set and T set. We also know the entire features and its directly parent set for each S set and T set variables.

```
public class STBayesNet extends QuasiBayesNet {
      ProbabilityVariable probability_variables_s[];
      ProbabilityFunction probability_functions_s[];
      ProbabilityVariable probability_variables_t[];
      ProbabilityFunction probability_functions_t[];
      int n_s;
      int n_t;
}
```

We defined class *STCBayesNet* for computing conditional causal effects $P_t(s|c)$ function. It inherits from class *SCBayesNet*. *STCBayesNet* can let function algorithm recognize C variable.

```
public class STCBayesNet  extends STBayesNet {
      ProbabilityVariable probability_variables_c[];
      ProbabilityFunction probability_functions_c[];
      int n_c;                }
```

We defined data structure Q representing Q[S], which is a basic factor, for representing value of causal effects in Bayesian networks. The quantity of Q[S] denotes the post-intervention distribution of S under an intervention of all other variables, that is, Q[S] = $P_{v\backslash s}(s)$ (see equation (6)).

```
public class Q {
        ProbabilityVariable[] Summation;
        Probability[] Probability;
}
```

Probability class was defined as a data structure for representing conditional probability. For example, a conditional probability P(a|b,c) is an object of Class Probability, "a" is an instance of name, {b, c} is an instance of evidence.

```
public class Probability {
        DiscreteVariable name;
        DiscreteVariable[] evidences;
        int num_evi;
        int num_value;
}
```

In our software system, the *main()* method is in Class *BayesianNetworks.JavaBayes.java*, where *BayesianNetworks* is a package and *JavaBayes* is a class. The identification functions are called from *JavaBayesInterface.EditorFrame. process_query()*, where *JavaBayesInterface* is a Package; *EditorFrame* is the Class by which the graph editor can be shown; *process_query()* is a method. The function call selections depend on a mode flag, named *mode_menu_choice*. For example, when *mode_menu_choice* is *IDENTIFY_H* (mode identify causal-effect algorithm explained in Chapter 3.1), method *print_ identifiability_analysis (PrintStream pstream, InferenceGraph ig)* is called from

method *process_query()*. Both of the methods are in class *EditorFrame*. Afterward, method *print_ identifiability_analysis (PrintStream pstream, InferenceGraph ig) of class EditorFrame* invokes method *print_identifiability_analysis(PrintStream out)* of Class *InferenceGraph*, where the system accesses identifying function components in the end. The process of method calling is described in Figure 11. The algorithms implemented in the system are described in previous chapter. The implementation codes will be listed in attachment A.

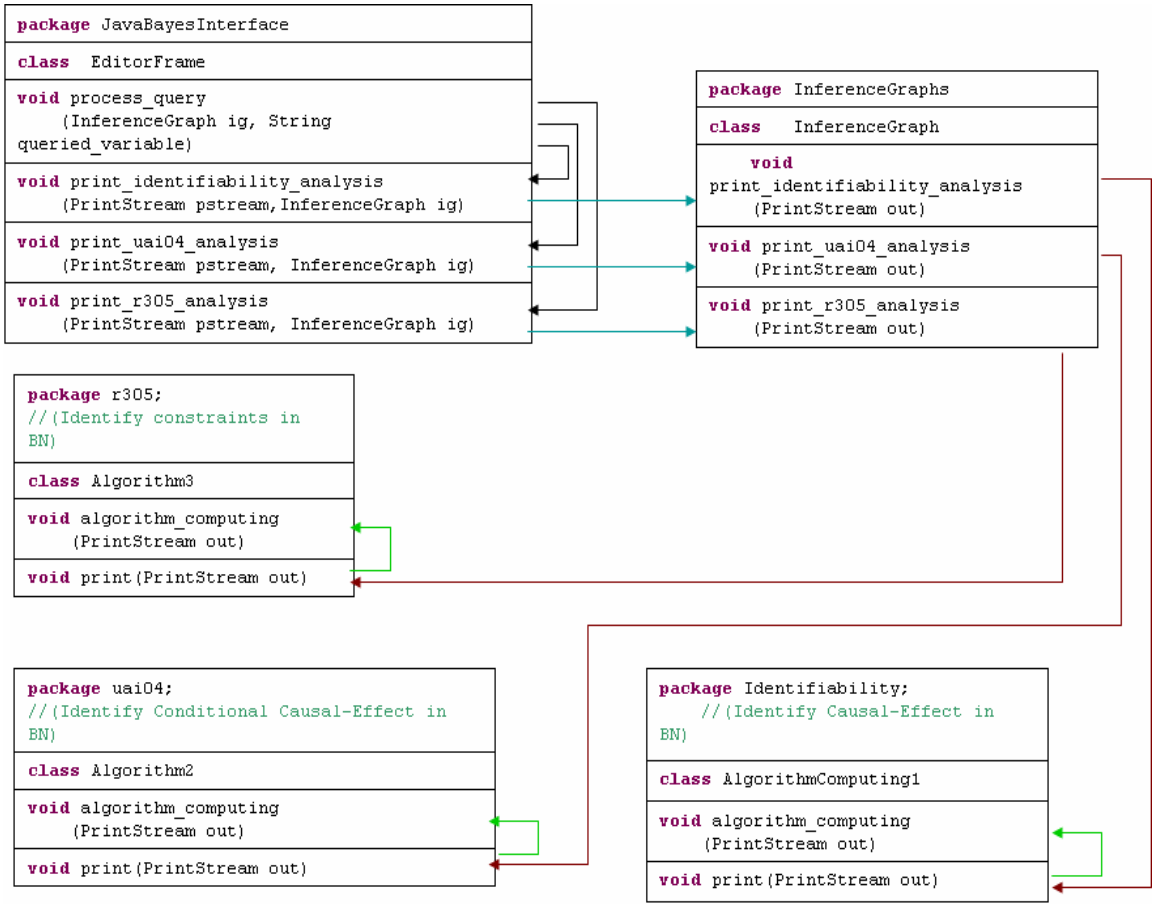



Figure 11. The process for call function algorithms

### 4.3.4 Creating Executable Jar File

A feature of Java2 is the ability to create executable jar file. A jar file is simply a file containing a collection of java .class files. To make a jar file executable, we need to specify where the "main" Class is in the jar file, so the "java" command knows what *main()* method to invoke in order to get the software going[27].

In the directory “Classes”, we created a file called "mainClass". This file contains a single line specifying where the main class is to be found in the jar file. Here is the single line:

Main-Class: BayesianNetworks.JavaBayes

Next, we create a jar file called Class.jar using the "jar" command in Java2. We use the "m" command line argument to specify the manifest file mainClass.txt, which adds information to the jar file on where the main class will be found. Here is the jar command:

>jar cmf mainClass Class.jar *.class

At last, we create windows batch file BN.bat. Below is the content of the batch file.

>java -classpath ./Classes BayesianNetworks.JavaBayes

We can run the software via double click BN.bat in a windows operating system.

# CHAPTER 5. SOFTWARE TESTING AND RESULT

We test our system with many representative test cases. The goal of the testing is to expose latent defects in the system. In this chapter, we display some of the test cases and compare their results with the system outputs.

Most test cases are generated from the related publications for ensuring they are correct and we tried to let the test cases can test the entire algorithm implementation of the system.

## 5.1 Test Cases for Identifying causal effects Pt(s)

In this section, we display three test cases to test the Identifying Causal-Effects component, Pt(s).

### Test Case 1

Consider the test case of identifying $P_{x_1 x_2}(y)$ in Figure 12. The case was studied by Pearl and Robins[23]. The causal BN graph G in Figure 12 is created by our software system.

G has two c-components $S_1$ = {X1, Z, Y} and $S_2$ = {X2}

The result of identifying $P_{x_1 x_2}(y)$ in G is:

$$P_{x_1 x_2}(y) = \sum_z P(y \mid x_1, x_2, z) P(z \mid x_1)$$

Figure 13 is the output of our software system.

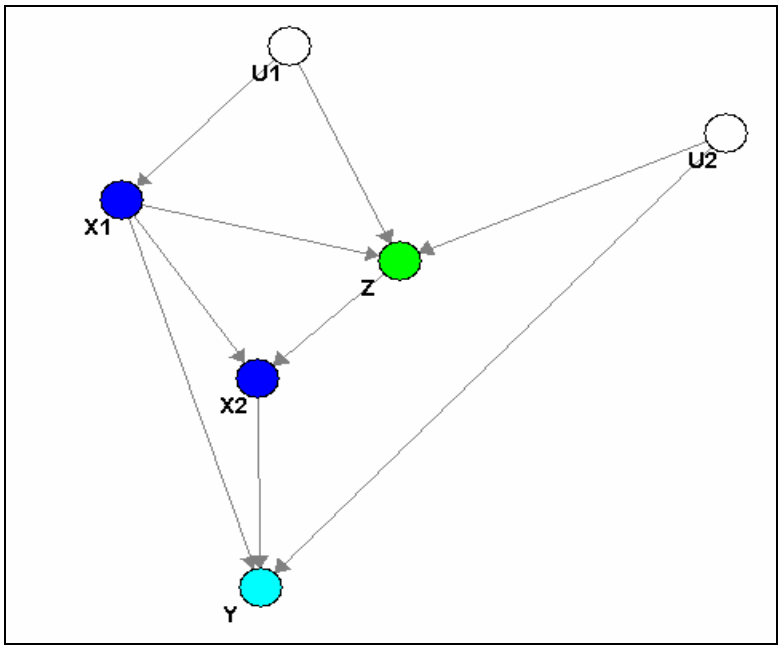



Figure 12. Causal Graph for test case 1.

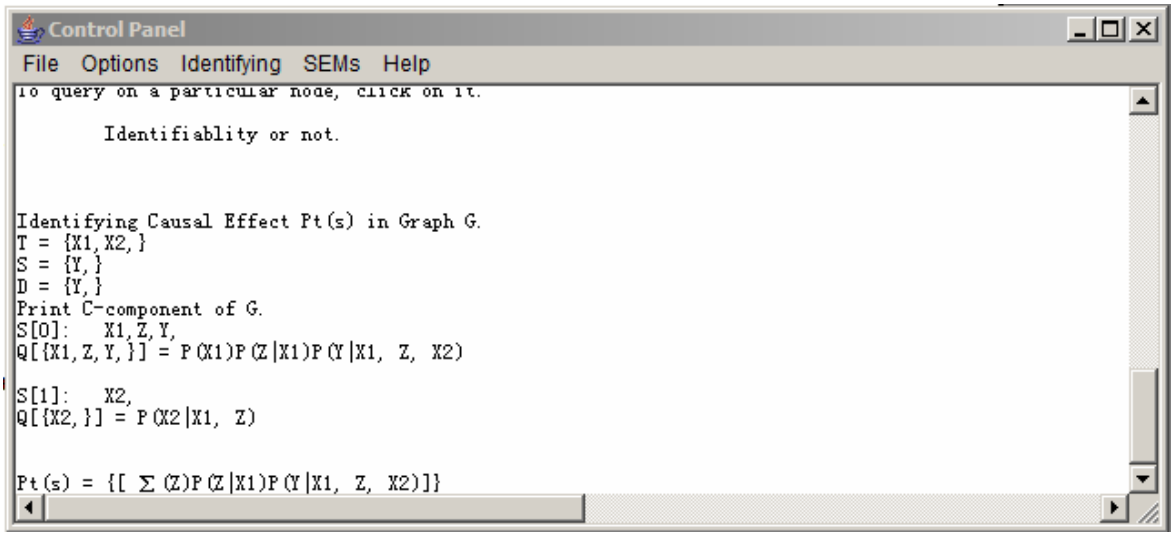



Figure 13. Our software system output for test case 1.

Obviously, the system output is same as the studied result.

## Test Case 2

Consider the test case of identifying $P_{x_1x_2}(y, z_1, z'_1)$ in Figure 14. The case was studied by Pearl and Robins too [23]. The causal BN graph G in Figure 14 is created by our software system.

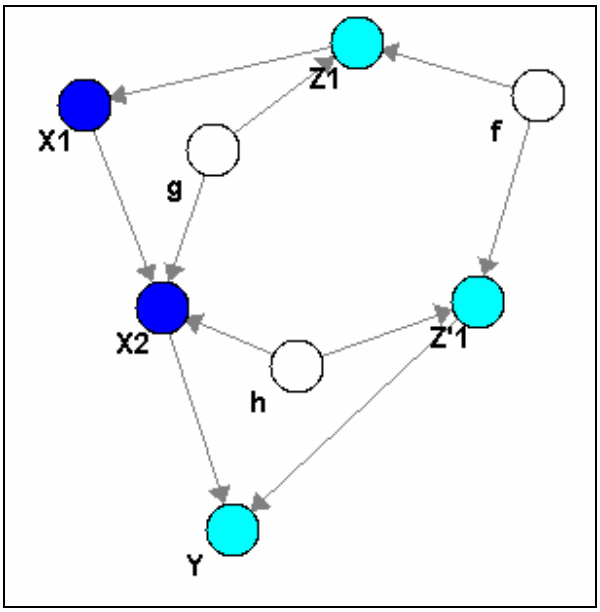



Figure 14. Causal Graph for test case 2.

G has three c-components $S_1$ = {X1 }, $S_2$ = {Y} and S3 = {X2, Z1, Z'1}.

The result of identifying $P_{x_1x_2}(y, z_1, z'_1)$ in G is Eq (19).

$$P_{x_1x_2}(y, z_1, z'_1) = P(y \mid x_2, z'_1)\sum_{x_2} Q[S3] = P(y \mid x_2, z'_1)P(z'_1 \mid x_1, z_1)P(z_1) \qquad (19)$$

Figure 15 is the output of our software system for test case 2.

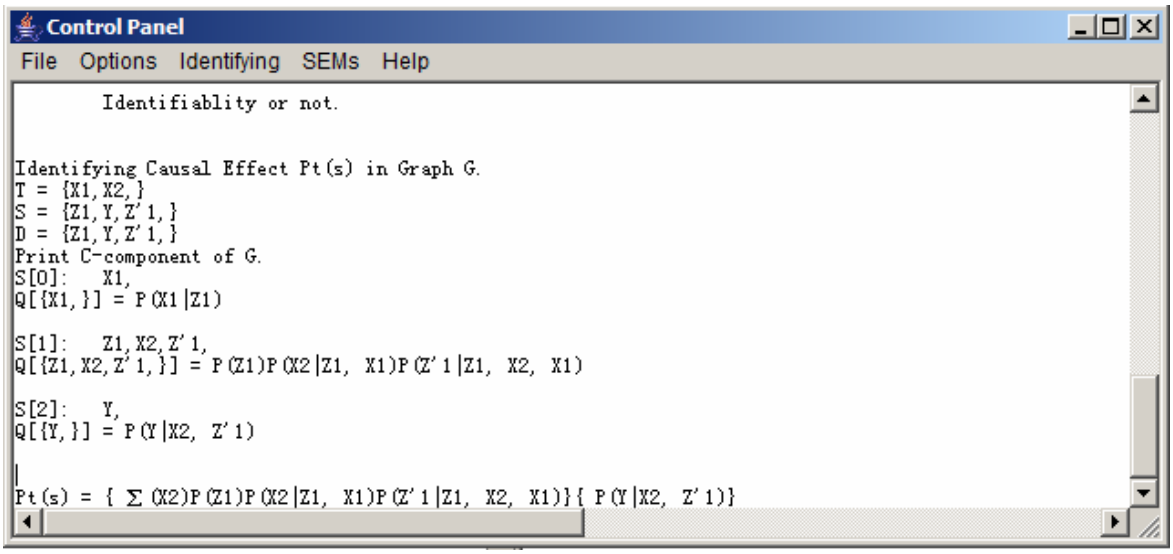



Figure 15. Our software system output for test case 2.

The system out show:

$$P_{x_1x_2}(y,z_1,z'_1)=\{\sum_{x_2}P(z_1)P(x_2\mid z_1,x_1)P(z'_1\mid z_1,x_2,x_1)\}\{P(y\mid x_2,z'_1)\} \tag{20}$$

Since we know $Q[S3]=P(z_1)P(x_2\mid z_1,x_1)P(z'_1\mid z_1,x_2,x_1)$ from Lemma 3, we can rewrite Eq (20) as $P_{x_1x_2}(y,z_1,z'_1)=P(y\mid x_2,z'_1)\sum_{x_2}Q[S3]$. Obviously, Eq (19) and Eq (20) have same value.

Hence, we have the conclusion that our system output has the same value with the studied result.

## Test Case 3

Consider the test case of identifying $P_{x_1x_2}(y)$ in Figure 16. The case was studied by Kuroki and Miyakawa [28]. The causal BN graph G in Figure 16 is created by our software system.

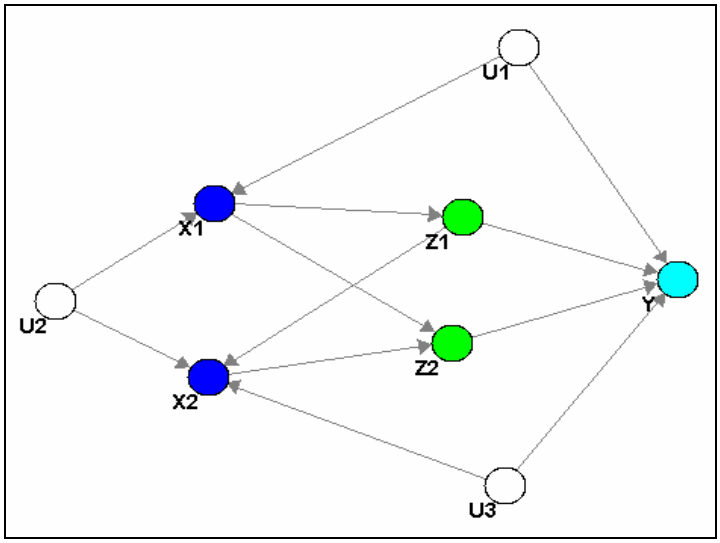



Figure 16. Causal Graph for test case 3.

G has three c-components $S_1$ = {X1, X2, Y}, $S_2$ = {Z1} and S3 = {Z2}.

The result of identifying $P_{x_1x_2}(y)$ in G is Eq (21).

$$P_{x_1,x_2}(y) = \{\sum_{z_1,z_2} P(z_1 \mid x_1)P(z_2 \mid x_1, x_2)\}\{\sum_{x_1,x_2} P(y \mid x_1, x_2, z_1, z_2)P(x_2 \mid x_1, z_1)P(x_1) \quad (21)$$

Figure 17 is the output of our software system for test case 3.

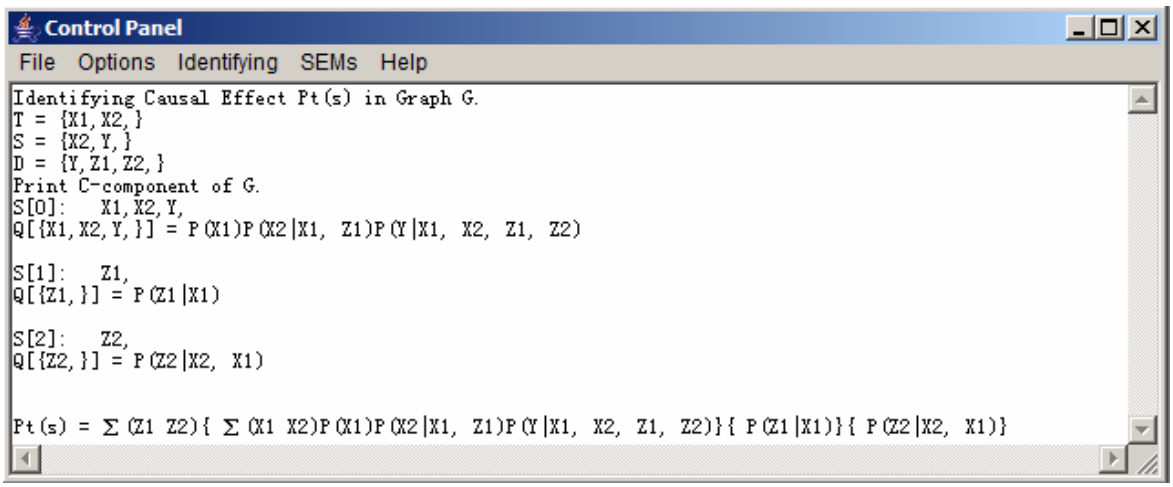



Figure 17. Our software system output for test case 3.

Obviously, the system output has the same accessed value as the studied result.

## 5.2 Test Cases for Identifying conditional causal effects Pt(s|c)

In this section, we display a test case which is used to test the Identifying Causal-Effects component, Pt(s) component.

Consider the test case of identifying $P_x(y \mid a)$ in Figure 18. The case was studied by Tian [20]. The causal BN graph G in Figure 18 is created by our software system.

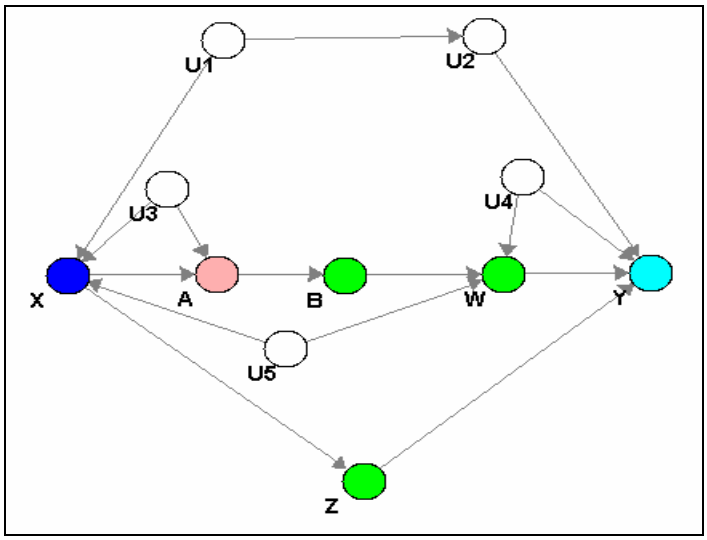



Figure 18. Causal Graph for test case 4.

G has three c-components $S_1$ = {B}, $S_2$ = {Z} and S3 = {X, A, W, Y}. By lemma 3, Q[{B}] = P(b|a), Q[{Z}] = P(z|x) and Q[S3] = P(y|z, w, b, a, x)P(w|b, a, x)P(a|x)P(x). The ancestors of Y and A in the subgraph with X removed are D = {A, B, W, Z, Y}. The c-component of the subgraph $G_D$ are {A}, {B}, {Z}, and {W, Y}.

Q[{B}] and Q[{Z}] are identifiable and are given above. Q[{W, Y}] can be computed by algorithm Identify({W,Y}, S3, Q[S3]) (shown in figure 5).

$$Q[\{W, Y\}] = \sum_{x,a} Q[S3]$$

Tian's study concludes that identifying $P_x(y \mid a)$ is identifiable and is given by Eq(22).

$$P_x(y \mid a) = \frac{\sum_{b,w,z} Q[\{B\}]Q[\{Z\}]Q[\{W,Y\}]}{\sum_{y,b,w,z} Q[\{B\}]Q[\{Z\}]Q[\{W,Y\}]} \tag{22}$$

Figure 19 is the output of our software system.

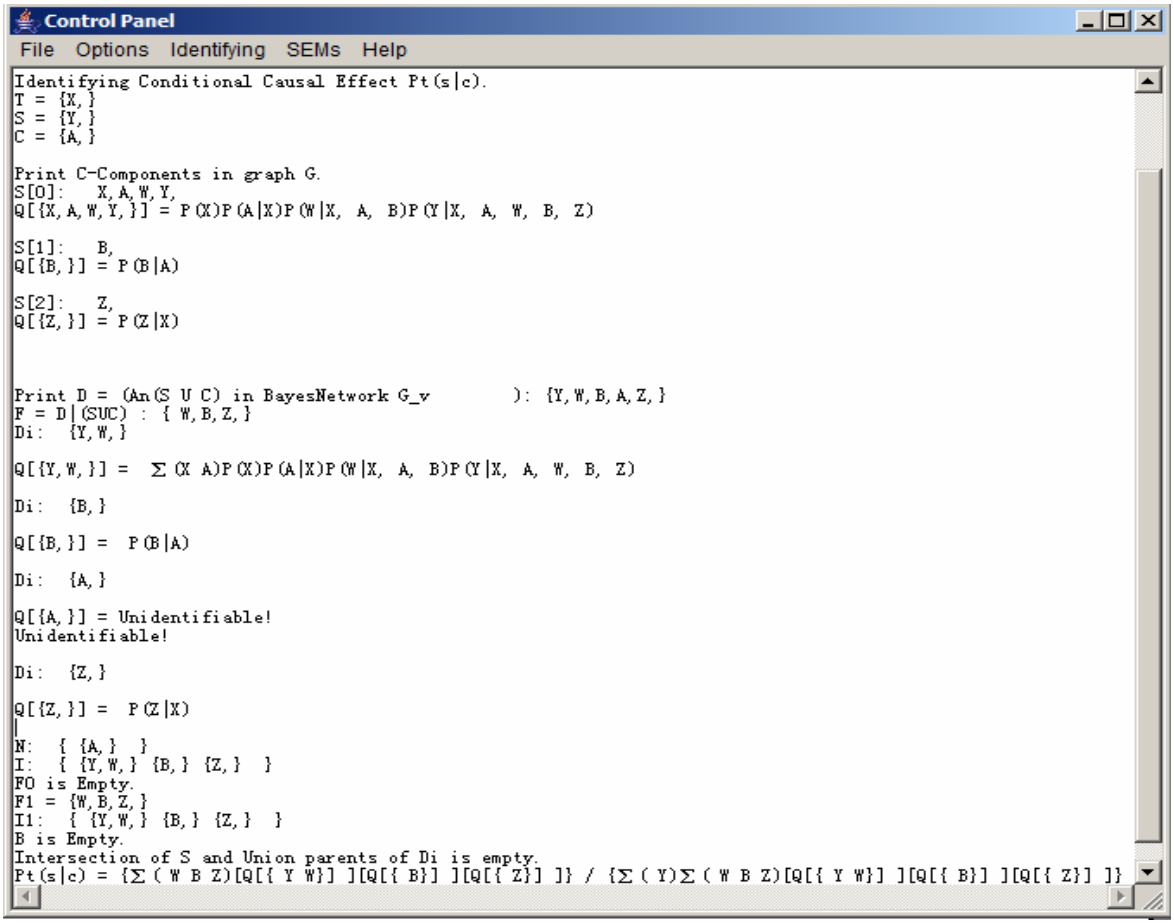



Figure 19. Our software system output for test case 4.

## 5.3 Test Cases for Identifying function constraints

In this section, we display a test case which is used to test the Identifying function constraints component.

Consider the test case of looking for constraints involving V1 to V5 in Figure 20. We can get the following result by algorithm described in section 3.3.

Computing Q[{V1}], Q[{V2}] and Q[{V3}] does not give any constraint.

$Q[\{V_4\}] = \sum_{v_2} P(v_4 \mid v_3, v_2, v_1) P(v_2 \mid v_1)$ implies a constraint on the distribution P(v) that the right hand side is independent of $v_1$.

V5 is in the c-component S={V1,V3,V5}. In the subgraph G(S), {V1},{V3} and {V1,V3} are the descendent set not containing V5.

$Q[S] = P(v_5 \mid v_4, v_3, v_2, v_1) P(v_3 \mid v_2, v_1) P(v_1)$ implies no constraints.

For case descendent set {V1}, section 3.3 algorithm gives $Q[\{V_5\}] = \frac{\sum_{v_1} P(v_5 \mid v_4, v_3, v_2, v_1) P(v_3 \mid v_2, v_1) P(v_1)}{\sum_{v_1} P(v_3 \mid v_2, v_1) P(v_1)}$. It implies a constraint that the right hand side is independent of $v_2$ and $v_3$.

For case descendent set {V3}, G({V1,V5}) can not be further partitioned into c-component. Algorithm cannot continue.

For case descendent set {V1,V3}, $Q[\{V_5\}] = \sum_{v_1, v_3} P(v_5 \mid v_4, v_3, v_2, v_1) P(v_3 \mid v_2, v_1) P(v_1)$ implies a constraint that the right hand side is independent of $v_2$.

The case was studied by Tian and Pearl [19]. The causal BN graph G in Figure 20 is created by our software system. Figure 21 is the output of our software system for. Obviously, the system output is same as the studied result.

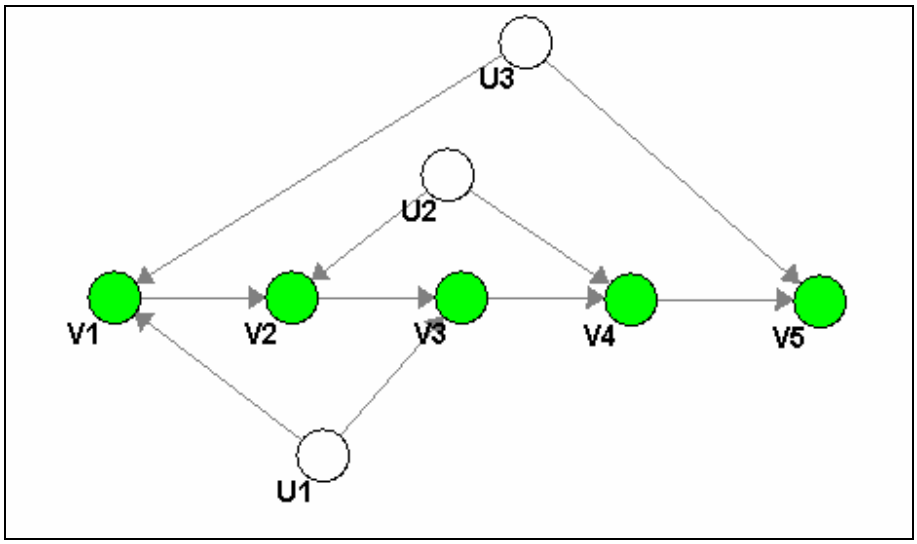



Figure 20. Causal Graph for test case 5.

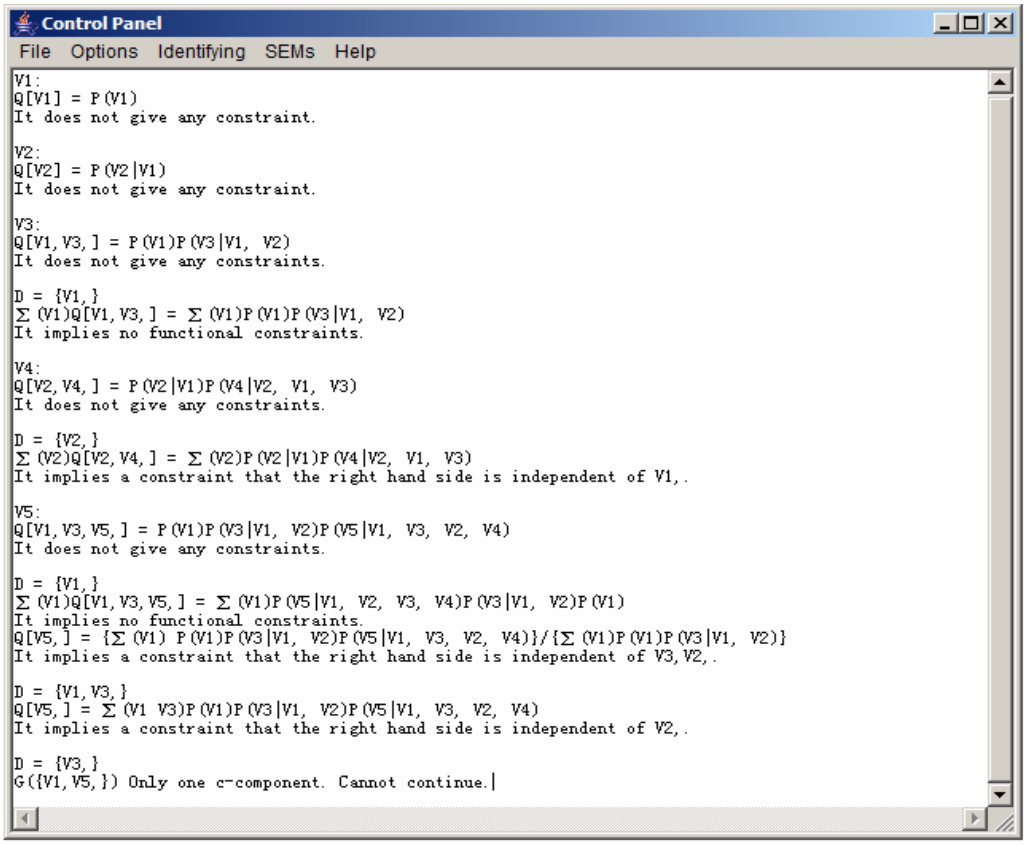



Figure 21. Our software system output for test case 5.

# CHAPTER 6 CONCLUSION AND FUTURE WORK

In this chapter, I will introduce my conclusion of my thesis and the future work that is helpful to improve the software.

## 6.1 Conclusion

Causal reasoning problems are critical for modeling the effects of actions and validate causal models. There are some other projects that use Java with Bayesian networks, such as JavaBayes produced by Carnegie Mellon University and BNJ developed by Kansas State University (KSU) KDD Lab. However, a few programming system implemented causal reasoning problems in causal Bayesian Networks.

In this thesis we describe a software system. The software system can solve some causal reasoning problems, such as, identify conditional and unconditional causal effects and find functional constraints in a given causal Bayesian Network with hidden variables. There are two user interfaces in the system. One can be used to create and display causal BN graph, the other one display the software use manual and analysis result of a given causal BN.

The presented design of the system meets the given requirements. The software system is reliable, easy extensible, have portability and reusability. Every functional component implementation follows the proved algorithms and lemmas that ensure the system give valid output. We tested the software system on windows OS. The testing analysis results show our software component implementation is correct and the output of the software match with the result from studied tested cases in some publications. This software is very helpful to researchers who are interested in Causal Bayesian Networks.

## 6.2 Future Work

The developed software system is to solve causal reasoning problems. Right now, we have three components, namely identify unconditional causal effects, identify conditional causal effects and find constraints in a causal Bayesian Networks with hidden variables. In the future, we would like to add more components to solve other causal reasoning problems, such as identifying causal-effects in a class of linear models and parameter identification in structural equation models.

We used Abstract Windows Toolkit (AWT) in our software development. Using AWT as Java GUI tool kit has some advantages as we mentioned in the thesis (4.3.1 Developing Environment). One of the advantages is that AWT lets our software has good compatibility with JavaBayes. However AWT is a very simple tool kit with limited GUI components, layout managers, and events. It is difficult to improve interface with AWT. On the other hand, Standard Widget Toolkit (SWT) is a low-level GUI tool kit comparable in concept to AWT. SWT makes the task of developing high quality user-interface easily. If development cost was enough, changing Java GUI tool kit to SWT would be a good choice to improve the user-interface.

# APPENDIX A. LEMMAS

**Lemma 1 [19]:** Let $W \subseteq C \subseteq V$. If W is an ancestral set in G(C) ($W = An^{v}(W)_{G(C)}$), or if $W'$ is a descendent set in G(C) ($W' = De^{v}(W')_{G(C)}$), then

$$\sum_{V_i \in C \setminus W} Q[C] = Q[W] \tag{7}$$

In Lemma 1, G(C) denotes the subgraph of G built only by variables in C and U(C). $W' = C \setminus W$.

And we always have $\sum_c Q[C] = 1$.

From lemma 1, we can calculate Q[W] from Q[C] by summing out C\W. The process is similar to marginalization in probability theory.

**Lemma 2 (Generalized Q-decomposition) [19]**: Let H $\subseteq$ N, and we have c-components from $H_1^{'},...,H_n^{'}$ in the sub graph GH, $H_i = H_i^{'} \cap H$, 1 $\leq$ i $\leq$ n, then

(i) Q[H] can be decomposed as

$$Q[H] = \prod_{i=1}^{n} Q[H_i] \tag{8}$$

(ii) Each Q[Hi] is computable from Q[H], in the following way. Let k be the number of variables in H, and let a topological order of variables in H be $V_{h_1} < ... < V_{h_k}$ in GH, Let $H^{(j)} = \{V_{h_1},...,V_{h_j}\}$ be the set of variables in H ordered before $V_{h_j}$ ( including $V_{h_j}$ ), j = 1, . . . , k, and $H^{(0)} = \phi$. Then each Q[$H_i$],i = 1, . . . , n, is given by

$$Q[H_i] = \prod_{\{j | V_{h_j} \in H_i\}} \frac{Q[H^{(j)}]}{Q[H^{(j-1)}]} \tag{9}$$

Where each Q[H(j)], j = 0, 1, . . . , k, is given by

$$Q[H^{(j)}] = \sum_{h \setminus h^{(j)}} Q[H] \tag{10}$$

In a special case of Lemma 2, H = V, Tian and Pearl give the following corollary (Lemma 3). All of unobserved variables are root nodes in the corollary [19].

**Lemma 3 (Q-decomposition) [20]:** Assuming that V is partitioned into c-components $S_1,\ldots, S_k$, we have

(i) $P(v) = \prod_i Q[S_i]$.

(ii) Each $Q[S_i]$ is computable from P(v). Let a topological order over V be $V_1 < \ldots < Vn$, and let $V^{(i)} = \{V_1,....,V_i\}$, i = 1,…,n, and $V^{(0)} = \phi$, Then each $Q[S_j]$, j = 1,…., k, is given by [20]

$$Q[S_j] = \prod_{\{i|V_i \in S_j\}} P(v_i \mid v^{(i-1)}) \tag{11}$$

(iii) Each factor $P(v_i \mid v^{(i-1)})$ can be expressed as

$$P(v_i \mid v^{(i-1)}) = P(v_i \mid pa^+(T_i) \setminus \{v_i\}) \tag{12}$$

Where $T_i$ is the c-component of $G(V^{(i)})$ that contains $V_i$.

We see that when all of the variables in the graph model are observed, each variable is independent of its non-descendants given its parents in the graph. When unobserved variables are involved, a variable is independent of its non-descendants given its effective parents, non-descendant variables in its c-component, and the effective parents of the non-descendant variables in its c-component [19].

# APPENDIX B. APPLICATION CODE EXAMPLES

## B.1 Identifiability in Causal Bayesian Networks (algorithms in section 3.2)

```
/**
 * algorithm_computing()
 * This is a implementation for algorithm in Paper "Identifiability
 * in Causal Bayesian Network". The paper was written by Huang and
 * Valtorta. The algorithms were studied by Tian and Pearl.
 * The S and T set should be set before use the function.
 * Both S and T are observable variables set.
 * the function will show whether a causal effect, that is,
 * the joint response of any sset S of variables to interventions
 * on a set T of action variables, denoted Pt(S) is identifiable or
 * not.
 */
public void algorithm_computing(PrintStream out){
      System.out.println("where start it?");
      pts = "==";
      STInference sti = new STInference(qbn, true);
      Probability[] P_T_S = null;
      Verma_Type_Functional_Constraint VTFC = new
                        Verma_Type_Functional_Constraint();
      Method M = new Method();
      methods m = new methods();
      //st = new STBayesNet(qbn);
      Enumeration e;
      int N_size = st.number_observed();
      int T_size = st.number_t();
      int S_size = st.number_s();
      ProbabilityVariable pv;
      ProbabilityVariable[] N = new ProbabilityVariable[N_size];
      ProbabilityVariable[] T = new ProbabilityVariable[T_size];
      ProbabilityVariable[] S = new ProbabilityVariable[S_size];
      N = st.get_observed_variables();
      T = st.get_probability_variables_t();
      S = st.get_probability_variables_s();
      Vector<ProbabilityVariable> N_T = new
                        Vector<ProbabilityVariable>();

      N_T = m.Subtraction(N,T);

      for (e = N_T.elements(); e.hasMoreElements(); ) {
      pv = (ProbabilityVariable)(e.nextElement());
      System.out.println("^N_T^"+pv.get_name());
      }
      Vector D = new Vector();
```

```
ProbabilityVariable [] N_t =
                  from_Vector_to_ProbabilityVariable(N_T);
Vector DUP = new Vector();
DUP = VTFC.DUP(qbn, N_t, out);
ProbabilityVariable[] dup = new
                  ProbabilityVariable[DUP.size()];
dup = m.from_Vector_to_ProbabilityVariable(DUP);
ProbabilityVariable[] union = new
                  ProbabilityVariable[DUP.size()+N_t.length];
union = m.Add(dup, N_t);
M.print_ProbabilityVariableArray(out, union);
BayesNet Gv_t = new BayesNet(VTFC.SubBayesNet(qbn, union,
                  "Gv_t"));
QuasiBayesNet q_Gv_t = new QuasiBayesNet(Gv_t);

D = m.Intersection(VTFC.ancestors(q_Gv_t, S), N);

////////////////////////////////////////////////
out.println("Identifying Causal Effect Pt(s) in Graph G.");
out.print("T = {");
String t = M.print_ProbabilityVariableArray(out, T);
out.println("}");
out.print("S = {");
String s = M.print_ProbabilityVariableArray(out, S);
out.println("}");
out.print("D = {");
M.print_Vector(out, D);
out.println("}");

QFraction Q_D[] = new QFraction[D.size()];

Q_D = Q(D, out);
if (Q(D, out) == null){
      out.print("Pt(s) is UNIDENTIFIABLE!");
      return;
}
else{

      out.print("Pt(s) = ");
      ProbabilityVariable[] d = new
            ProbabilityVariable[D.size()];
      d=from_Vector_to_ProbabilityVariable(D);
      Vector<ProbabilityVariable> Ds = new
            Vector<ProbabilityVariable>();
      ProbabilityVariable[] D_S = new
            ProbabilityVariable[D.size()-S_size];
      Ds = Subtraction(d, S);
      D_S = from_Vector_to_ProbabilityVariable(Ds);
      if (D_S.length==0){
            for (int i=0; i<Q_D.length; i++){
                  out.print("{");
                  out.print(QFprint(Q_D[i],out));
                  out.print("}");
            }
      }
```

```
                else{
                        String R = "∑(";
                        for (int i=0; i<D_S.length; i++){
                                if(i==D_S.length-1){
                                        R=R+D_S[i].get_name()+")";
                                }
                                else
                                        R=R+D_S[i].get_name()+" ";
                        }
                        System.out.print(R);
                        out.print(R);
                        for (int i=0; i<Q_D.length; i++){
                                if(Q_D[i]!=null){
                                        out.print("{");
                                        System.out.print("{");
                                        out.print(QFprint(Q_D[i],out));
                                        out.print("}");
                                        System.out.print("}");
                                }
                        }
                }

        }
        return ;

        }
```

## B.2 Identifying Conditional Causal Effect (algorithms in section 3.2)

```
    /**
     * algorithm_computing()
     * This is a implementation for algorithm in Paper "Identifying
     * Conditional Causal Effects". The paper was written by Tian
     * . The algorithms were also studied by Tian.
     * The S, T and C set should be set before use the function.
     * The S, T and C are observable variables set, and they are
distinct.
     * the function will show whether a causal effect, that is,
     * the joint response of any set S of variables to interventions
     * on a set T of action variables and conditioned on another set C,
     * denoted Pt(s|c) is identifiable or not.
     */

    public void algorithm_computing(PrintStream out){

        STCInference stcI = new STCInference(qbn, true);
        AlgorithmComputing Huang = new AlgorithmComputing(stc);
        Method thisM = new Method();
        methods Method = new methods();
        Verma_Type_Functional_Constraint VTFC = new
```

```
            Verma_Type_Functional_Constraint();
int size = stc.number_observed();
int U_size = stc.number_unobserved();
int all_size = size+U_size;
ProbabilityVariable[] All= new ProbabilityVariable[all_size];
Vector[] G_C_Com = new Vector[all_size];

All = stc.get_probability_variables();
/***********/
/* Phase 1 */
/***********/

/* Find c-components of G */
G_C_Com = Huang.partition_c_component(All);
Q[] Qs = new Q[G_C_Com.length];
//Qs = print_C_Components(out, G_C_Com, stcI);

ProbabilityVariable[] S = new
            ProbabilityVariable[stc.number_s()];
ProbabilityVariable[] C = new
            ProbabilityVariable[stc.number_c()];
ProbabilityVariable[] S_U_C = new
            ProbabilityVariable[stc.number_c()+stc.number_s()];

S = stc.get_probability_variables_s();
C = stc.get_probability_variables_c();

/*S Union C*/
S_U_C = Method.Add(S, C);
Vector G_V_T = new Vector();
ProbabilityVariable[] V = new
      ProbabilityVariable[stc.number_observed()];
ProbabilityVariable[] T = new
      ProbabilityVariable[stc.number_t()];
Vector<ProbabilityVariable> V_T = new
      Vector<ProbabilityVariable>();
V = stc.get_observed_variables();
T = stc.get_probability_variables_t();
/* V_T =  V\T  */
V_T = Method.Subtraction(V, T);
/* G_v_t = Gv\t  That is a BayesNetwork, where include
variables in v\t*/
G_V_T = Huang.Gc(V_T);
QuasiBayesNet G_v_t = new QuasiBayesNet();
G_v_t = Huang.from_Vector_to_QuasiBayesNet(G_V_T, "Gv_t");
/* D = An(S U C) in BayesNetwork G_v_t */

for (int k=0; k<G_v_t.number_variables(); k++){
      if (G_v_t.get_probability_variable(k).is_observed())
}


int An_len = VTFC.ancestors(G_v_t, S_U_C).length;
ProbabilityVariable[] An_SC = new ProbabilityVariable[An_len];
An_SC = VTFC.ancestors(G_v_t, S_U_C);
```

```
int An_i = 0;
ProbabilityVariable[] An = new ProbabilityVariable[An_len];
for (int k=0;k<An_len; k++){
      if (An_SC[k].is_observed()){
            An[An_i] = An_SC[k];
            An_i ++;
      }
}
ProbabilityVariable[] D = new ProbabilityVariable[An_i];
for(int k = 0; k< An_i; k++){
      D[k] = An[k];
}
/*  subgraphh Gd */
Vector DUP = new Vector();
DUP = VTFC.DUP(qbn, D, out);
ProbabilityVariable[] dup = new
            ProbabilityVariable[DUP.size()];
dup = Method.from_Vector_to_ProbabilityVariable(DUP);
ProbabilityVariable[] union1 = new
            ProbabilityVariable[DUP.size()+D.length];
union1 = Method.Add(dup, D);
BayesNet Gd = new BayesNet(VTFC.SubBayesNet(qbn, union1,
            "Gd"));
QuasiBayesNet q_Gd = new QuasiBayesNet(Gd);
Vector[] G_D_Com = new Vector[Gd.number_variables()];
AlgorithmComputing AC = new AlgorithmComputing(q_Gd);
G_D_Com = AC.partition_c_component(D);
int num_c = G_D_Com.length;
out.println("Identifying Conditional Causal Effect Pt(s|c).");
out.print("T = {");
String t = thisM.print_ProbabilityVariableArray(out, T);
out.println("}");
out.print("S = {");
String s = thisM.print_ProbabilityVariableArray(out, S);
out.println("}");
out.print("C = {");
String c = thisM.print_ProbabilityVariableArray(out, C);
out.println("}");
out.println();
out.println("Print C-Components in graph G.");
Qs = print_C_Components(out, G_C_Com, stcI);
out.println();
out.print('\n'+"Print D = (An(S U C) in BayesNetwork G_v\t):
            {");
thisM.print_ProbabilityVariableArray(out, D);
out.println("}");
/* F = D\(S U C) */
Vector<ProbabilityVariable> VF = new
      Vector<ProbabilityVariable>();
ProbabilityVariable[] F = new ProbabilityVariable[D.length -
      S_U_C.length];
VF = Method.Subtraction(D, S_U_C);
F = Huang.from_Vector_to_ProbabilityVariable(VF);
out.print("F = D|(SUC) : { ");
thisM.print_ProbabilityVariableArray(out, F);
```

```
out.println("}");
/*******************/
/*    Phase 2      */
/*******************/

Vector N = new Vector();
Vector I = new Vector();
Vector I0 = new Vector();
Vector I1 = new Vector();
ProbabilityVariable[] F_0;
ProbabilityVariable[] F_1;
QFraction[] QF = new QFraction[G_D_Com.length];
QFraction[] QF_out = new QFraction[G_D_Com.length];
QF_out = Phase2_1(out, N, I, QF, G_D_Com, G_C_Com, Qs);
Enumeration g, e;
out.print("N:  { ");
for (g = N.elements(); g.hasMoreElements(); ) {
      ProbabilityVariable[] PVs_N = (ProbabilityVariable[])
            g.nextElement();
      out.print("{");
      thisM.print_ProbabilityVariableArray(out, PVs_N);
      out.print("} ");
}
out.println(" } ");

out.print("I:  { ");
for (g = I.elements(); g.hasMoreElements(); ) {
      ProbabilityVariable[] PVs_I = (ProbabilityVariable[])
            g.nextElement();
      out.print("{");
      thisM.print_ProbabilityVariableArray(out, PVs_I);
      out.print("} ");
}
out.println(" } ");
//Phase2_2
if (N.size()==0){
      String R = output(out, QF_out,S, F, I, G_D_Com);
      out.println(R);
      return;
}
else {
      /***********/
      /* Phase 3 */
      /***********/

      /*Initialize F0 = F n (To all of Di in N Pa(Di)) */
      ProbabilityVariable[] Union_pa = null;
      for (g = N.elements(); g.hasMoreElements(); ) {
            ProbabilityVariable[] PVs_N =
      (ProbabilityVariable[]) g.nextElement();
            ProbabilityVariable[] PVs_N_pa =
                  VTFC.ancestors(qbn, PVs_N);
            Union_pa = Method.Add(Union_pa, PVs_N_pa);

      }
```

```
ProbabilityVariable[] Union_parents = null;
if (Union_pa != null){
  for(int i=0; i<Union_pa.length; i++){
      if (Union_pa[i].is_observed())
            Union_parents = Method.Add(Union_parents,
                        Union_pa[i]);
  }
}
Vector<ProbabilityVariable> F0 = new
      Vector<ProbabilityVariable>();
Vector<ProbabilityVariable> F1 = new
      Vector<ProbabilityVariable>();
F0 = Method.Intersection(F, Union_parents);
if (F0.size()==0){
      out.println("F0 is Empty.");
}
else{
      out.print("F0 = ");
      thisM.print_Vector(out, F0);
      out.println("} ");
}
F_0 = new ProbabilityVariable[F0.size()];
F_0 = Method.from_Vector_to_ProbabilityVariable(F0);
F_1 = new ProbabilityVariable[F.length-F0.size()];
/* F1 = F\F0 */
F1 = Method.Subtraction(F, F_0);
F_1 = Method.from_Vector_to_ProbabilityVariable(F1);
out.print("F1 = {");
thisM.print_ProbabilityVariableArray(out, F_1);
out.println("} ");
/* I0 = empty; I1 = I*/
I0 = new Vector();
I1 = I;
// Print I1
out.print("I1:  { ");
for (g = I1.elements(); g.hasMoreElements(); ) {
      ProbabilityVariable[] PVs_I1 =
            (ProbabilityVariable[]) g.nextElement();
      out.print("{");
      thisM.print_ProbabilityVariableArray(out, PVs_I1);
      out.print("} ");
}
out.println(" } ");
boolean stop_Phase3_step2 = true;
while (stop_Phase3_step2){

Vector I1_copy = I1;
for (g = I1.elements(); g.hasMoreElements(); ) {
      ProbabilityVariable[] PVs_I1 =
            (ProbabilityVariable[]) g.nextElement();
      ProbabilityVariable[] Di_pa = VTFC.ancestors(qbn,
                  PVs_I1);
      Vector Intersection = Method.Intersection(Di_pa,
                  F_0);
      if (Intersection.size()>0){
```

```
                I1_copy.remove(PVs_I1);
                I0.add(PVs_I1);
          }
    }
    I1 = I1_copy;
    ProbabilityVariable[] Union_pa_B = null;
    for (g = I0.elements(); g.hasMoreElements(); ) {
          ProbabilityVariable[] PVs_I0 =
                (ProbabilityVariable[]) g.nextElement();
          ProbabilityVariable[] PVs_I0_pa =
                      VTFC.ancestors(qbn, PVs_I0);
          Union_pa_B = Method.Add(Union_pa_B, PVs_I0_pa);

    }
    int Union_size;
    if (Union_pa_B == null){
    Union_size = 0 ;
    }
    else{
          Union_size = Union_pa_B.length;
    }
    ProbabilityVariable[] Union_parents_B_tem = new
                ProbabilityVariable[Union_size];
    int idx_b =0;
    if(Union_pa_B != null){
      for(int i=0; i<Union_pa_B.length; i++){
          if (Union_pa_B[i].is_observed()){
                Union_parents_B_tem[idx_b] = Union_pa_B[i];
              idx_b++;
          }
      }
    }
    ProbabilityVariable[] Union_parents_B = new
                      ProbabilityVariable[idx_b];
    for(int i=0; i<idx_b; i++){
          Union_parents_B[i] = Union_parents_B_tem[i];
      }

    Vector<ProbabilityVariable> B = new
                      Vector<ProbabilityVariable>();
B = Method.Intersection(F_1, Union_parents_B);
// Print B
if (B==null){
    out.println("B is Empty.");
    stop_Phase3_step2 = false;
}
else if (B.size()==0){
    out.println("B is Empty.");
    stop_Phase3_step2 = false;
}
else{
    Vector F1_copy = F1;
    Vector F0_copy = F0;
    for (g = B.elements(); g.hasMoreElements(); ) {
                ProbabilityVariable pv_B =
```

```
                        (ProbabilityVariable) g.nextElement();
                    F1_copy.remove(pv_B);
                    F0_copy.add(pv_B);
                }
            F1 = F1_copy;
            F0 = F0_copy;
            F_1 = Method.from_Vector_to_ProbabilityVariable(F1);
      }// end else if B not null
        }//end while (stop_Phase3_step2)
}//end the else of Phase 3
/***********/
/* Phase 4 */
/***********/
ProbabilityVariable[] Union_pa = null;
for (g = N.elements(); g.hasMoreElements(); ) {
      ProbabilityVariable[] PVs_N = (ProbabilityVariable[])
                                    g.nextElement();
      ProbabilityVariable[] PVs_N_pa = VTFC.ancestors(qbn,
                                    PVs_N);
      Union_pa = Method.Add(Union_pa, PVs_N_pa);

}
for (g = I0.elements(); g.hasMoreElements(); ) {
      ProbabilityVariable[] PVs_I0 = (ProbabilityVariable[])
                                    g.nextElement();
      ProbabilityVariable[] PVs_I0_pa = VTFC.ancestors(qbn,
                                    PVs_I0);
      Union_pa = Method.Add(Union_pa, PVs_I0_pa);

}

ProbabilityVariable[] Union_parents = null;
if (Union_pa != null){
  for(int i=0; i<Union_pa.length; i++){
      if (Union_pa[i].is_observed())
            Union_parents = Method.Add(Union_parents,
                                    Union_pa[i]);
  }
}
Vector<ProbabilityVariable> phase4 = new
                  Vector<ProbabilityVariable>();
phase4 = Method.Intersection(S, Union_parents);
if (phase4 == null){
      out.println("Intersection of S and Union parents of Di
                  is empty.");
      out.println();
      out.print("Pt(s|c) = ");
      String R = output(out, QF_out,S, F_1, I1, G_D_Com);
      out.println(R);
}
else if (phase4.size()==0){
      out.println("Intersection of S and Union parents of Di
                  is empty.");
      out.print("Pt(s|c) = ");
      String R = output(out, QF_out,S, F_1, I1, G_D_Com);
```

```
            out.println(R);
        }
        else{
            out.print("Pt(s|c) is unidentifiable ");
        }
        return;
}
```

## B.3 Identifying Functional Constraints (algorithms in section 3.3)

```
/**
 * Identifying_Constraints()
 * This is a implementation for algorithm in Paper "On the
 * Implications of Causal Models with Hidden Variables". The
 * paper was written by Tian and Pearl.
 * The algorithms were also studied by Tian and Pearl.
 */
public void Identifying_Constraints(QuasiBayesNet qbn, PrintStream
                    out){
        methods m = new methods();
        Method M = new Method();
        STCBayesNet STC = new STCBayesNet(qbn);
        STCInference STCI = new STCInference(qbn, true);
        int len = STC.number_observed();
        ProbabilityVariable[] V = new ProbabilityVariable[len];
        V = STC.get_observed_variables();
        V = STCI.TopologicalOrder(V);
        ProbabilityVariable[] Si = new ProbabilityVariable[len];
        Q Qs = new Q();
        String Qprint = "";
        String Q_variables = "";
        for(int i=0; i<len; i++){
            if (i==1){
                out.println("Q[" +V[0].get_name() +"] = P(" +
                    V[0].get_name() +")");
                out.println("Q[" +V[0].get_name() +"] does not
                    give any constraint.");
                out.println();
            }
            else{
            /* (A1) */
            /*  subgraphh G(Vi) */
            out.println(V[i].get_name()+":");
            //i=2;
            ProbabilityVariable[] Vi = new ProbabilityVariable[i+1];
            Vi =get_Vi(V, i);
            Vector DUP = new Vector();
            DUP = DUP(qbn, Vi, out);
            ProbabilityVariable[] dup = new
                ProbabilityVariable[DUP.size()];
            dup = m.from_Vector_to_ProbabilityVariable(DUP);
            ProbabilityVariable[] union = new
                ProbabilityVariable[DUP.size()+Vi.length];
```

```
union = m.Add(dup, Vi);
BayesNet Gvi = new BayesNet(SubBayesNet(qbn, union,
                  "Gvi"));
QuasiBayesNet q_Gvi = new QuasiBayesNet(Gvi);
//LinkedList com = new LinkedList();
Vector[] com = new Vector[i+1];
AlgorithmComputing AC = new AlgorithmComputing(q_Gvi);
com = AC.partition_c_component(Vi);
int num_c = com.length;
System.out.println("&&&&&"+com.length);
int Si_idx = -1;
if (num_c == 1){
      Q_variables = "";
      for (int k=0; k<Vi.length; k++){
            Q_variables += Vi[k].get_name() +",";
      }
      out.print("Q[" +Q_variables + "] ");
      out.println("Only one c-component. Cannot
                  continue.");
      out.println();
}
if(num_c >1){
      /* A1 */
      Si_idx = find_Si(com, V[i]);
      Q_variables = "";
      Si =
      m.from_Vector_to_ProbabilityVariable(com[Si_idx]);
      Qs = Corollary_1(qbn, Si,out);
      for (int k=0; k<Qs.number_Probability(); k++){
                  Q_variables +=
      Qs.getProbability()[k].getName().get_name() +",";
      }
      Qprint = AC.Qprint(Qs);
      out.print("Q[" +Q_variables + "] = " + Qprint);
      out.print('\n');
      if (Si.length == 1){
            out.println("Q[" + Q_variables + "] does not
            give any constraints.");
            out.print('\n');
      }
      else{
      boolean flag1 = true;
      while(flag1){
      /* (A2) */
      /* subgraph G(Si)*/
      DUP = DUP(qbn, Si, out);
      dup = m.from_Vector_to_ProbabilityVariable(DUP);
      union = m.Add(dup, Si);
     Gvi = new BayesNet(SubBayesNet(qbn, union, "Gsi"));
      q_Gvi = new QuasiBayesNet(Gvi);
      System.out.println("Gsi:");
      for (int k=0; k<q_Gvi.number_variables(); k++){
      ProbabilityVariable[] Si_Vi = new
            ProbabilityVariable[Si.length-1];
      /* each descendent set D in G(Si), that not
```

```
      contain Vi */
Si_Vi = m.Subtraction(Si, V[i]);
Vector subset = new Vector();
Combine(Si_Vi, subset);
//subset = getSubset(Si_Vi);
Enumeration e;
int number_D = subset.size();
int D_Counter =0;
for (e = subset.elements(); e.hasMoreElements(); )
{     ProbabilityVariable[] pv = new
            ProbabilityVariable[i];
pv = (ProbabilityVariable[])(e.nextElement());
out.print("D = {"); //pv is D of r305 algorithm.
      for (int j=0; j<pv.length; j++){
            out.print(pv[j].get_name()+",");
      }
      out.print("}  ");
      int de_size = descendent(q_Gvi,
            pv,out).length;
      ProbabilityVariable[] de = new
            ProbabilityVariable[de_size];
      de  = descendent(q_Gvi, pv,out);
      if(!equle (pv, de)){
            D_Counter ++;
            out.println("is not a descendent set.
            ");
      }
      else{
           System.out.println("is descendentral.
            ");
ProbabilityVariable[] constraint = new
            ProbabilityVariable[len];
Vector Constraint = new Vector();
int[] constraint_len = new int[1];
Q Qsi_d = new Q();
Vector<ProbabilityVariable> W_p = new
            Vector<ProbabilityVariable>();
W_p = m.Subtraction(Si, pv);
ProbabilityVariable[] D_P = new
      ProbabilityVariable[W_p.size()];
D_P = m.from_Vector_to_ProbabilityVariable(W_p);
Qsi_d = Lemma1 (qbn, q_Gvi,D_P, Si, Qs, Constraint,
            constraint_len, out);
Vector t1 = new Vector();
      if(Qsi_d !=null){
            //ProbabilityVariable[] tem2 = new
            ProbabilityVariable[c_len];

            t1 = m.Subtraction(Paplus(qbn,Si,out),
                  pv);
            ProbabilityVariable[] tem1 = new
                  ProbabilityVariable[t1.size()];
            tem1 =
          m.from_Vector_to_ProbabilityVariable(t1);
            //tem2 = Paplus(qbn,W_P,out);
```

```
        Constraint =m.Subtraction(tem1,
              Paplus(qbn,D_P,out));
        Enumeration ff;
        ProbabilityVariable xxx = new
              ProbabilityVariable();
        for (ff = Constraint.elements();
              ff.hasMoreElements(); ) {
              xxx =
    (ProbabilityVariable)(ff.nextElement());}
        if(D_P.length==1){
              flag1 = false;
              Q_variables = "";
        for(int q=0; q<D_P.length; q++){
              Q_variables  +=  D_P[q].get_name()
              +",";
        }
        out.print("Q[" +Q_variables + "] = " +
        AC.Qprint(Qsi_d));
        out.print('\n');
        if (Constraint.size()==0){

             out.print("Q[" + Q_variables + "]
              does not give any constraints.");
              out.print('\n');
              out.print('\n');
        }
        else{

              out.println("Q[" + Q_variables +
              "] It implies a constraint on
              P(v) that the right hand side is
              independent of " +
              M.print_Vector(out, Constraint)
              + ".");
              out.print('\n');
        }
        } // if(D_P.length==1)
        if (D_P.length>1){
        /* subgraph G(D_pi)*/
        DUP = DUP(qbn, D_P, out);
        dup =
m.from_Vector_to_ProbabilityVariable(DUP);
        union = m.Add(dup, D_P);
        Gvi = new BayesNet(SubBayesNet(qbn,
        union, "GDpi"));
        QuasiBayesNet q_Gvi_2 = new
              QuasiBayesNet(Gvi);
        ProbabilityVariable[] Ei = new
        ProbabilityVariable[D_P.length];
        AlgorithmComputing AC2 = new
        AlgorithmComputing(q_Gvi_2);
        com = AC2.partition_c_component(D_P);
        num_c = com.length;
        Si_idx = -1;
        if (num_c>1){
```

```
        Si_idx = find_Si(com, V[i]);
        Ei =
        m.from_Vector_to_ProbabilityVari
              able(com[Si_idx]);
      D_P  =  STCI.TopologicalOrder(D_P);
        Q[] QDP = new Q[1];
        QDP[0] = Qsi_d;
        QFraction QDpi = new
        QFraction(QDP);
        QFraction QEi = new QFraction();
        QEi = AC.lemma2(QDpi, Ei, D_P);

        Constraint =
        m.Subtraction(Paplus(qbn, D_P,
        out), Paplus(qbn, Ei, out));
        Q_variables = "";
        for(int q=0; q<Ei.length; q++){
               Q_variables +=
        Ei[q].get_name() +",";
        }
        out.print("Q[" +Q_variables + "]
        = " );
        AC.QFprint(QEi, out);
        out.print('\n');
        if (Constraint.size()>0){
              out.println("Q[" +
              Q_variables + "] It
              implies a constraint on
              P(v) that the right hand
              side is independent of " +
              M.print_Vector(out,
              Constraint) + ".");
              out.print('\n');
        }
        else{
              out.print("Q[" +
                    Q_variables + "]
              does not give any
              constraints.");
              out.print('\n');
              out.print('\n');
        }
        if (Ei.length < 2){

              flag1 = false;
        }
}// if (num_c>1)
else if (num_c==1){
        flag1 = false;
        Q_variables = "";
        for(int q=0; q<D_P.length; q++){
Q_variables  +=  D_P[q].get_name()  +",";
        }
out.print("Q[" +Q_variables + "] ");
```

```
                                    out.println("Only one c-component.
                                          Cannot continue.");
                                    } //else if (num_c==1)   //else ---
                                    } //end if (D_P.length>0)
                              }//if(Qsi_d !=null)
                              }//end if ancestral or descendent
                        }//end for
                        if (D_Counter == number_D){
                              flag1 = false;
                        }
                        }//while

                  }//if (Si.length == 1) else{}/*A2*/
                  }//end if c_num>1
            }//else /*A1*/
      }
      return;
}
```

# ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis.

First, I wish to thank Dr. Jin Tian for supervising and instructing this research and this thesis and for providing lots of ideas and feedback for the thesis. I would like to thank my committee members for their efforts and contributions to this work: Dr. Shashi K Gadia and Dr. Doug Jacobson. Also, I would also like to extend my gratitude to Fang Peng, Jia Tao and Changsung Kang for their comments and suggestions that eventually helped me finalize the work. Finally, I would like to thank my family, especially my husband, for their invaluable help during the university years and during my life.

# REFERENCE


1. Heckerman, D., and Shachter, R. 1995. Decision-theoretic foundations for causal reasoning. *Journal of Artificial Intelligence Research,* **3**: 405–430.
2. Cozman, F.G., *JavaBayes - version 0.346*. 1998 - 2001, http://www.cs.cmu.edu/~javabayes/Home/
3. Niedermayer, D. 1998. *An Introduction to Bayesian Networks and their Contemporary Applications*. http://www.niedermayer.ca/papers/bayesian/index.html.
4. Tian, J. 2004 Identifying Linear Causal Effects. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-04)*, 104-110.
5. Cheeseman, P. 1985. In Defense of Probability. In *the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, 1002-1009.
6. Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
7. Russell, S. and Norvig, P. 2006. *Artificial Intelligence: A modern approach*. 2ed. 492-533.
8. Nadkarni, S., and Shenoy, P. 2004. A causal mapping approach to constructing Bayesian networks. *Decision Support Systems,* 38(2): 259-281.
9. Speigelhalter, J., Lauritzen, L. and Cowell, G. 1993. Bayesian analysis in expert systems. *Statistical Science,* 8(3): 219-247.
10. Murphy, K. 1998. A Brief Introduction to Graphical Models and Bayesian Networks. http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html.
11. Charles River Analytics, Inc. 2004 About Bayesian Belief Networks. http://www.cra.com/pdf/BNetBuilderBackground.pdf.
12. Bayesian network. http://en.wikipedia.org/wiki/Bayesian_network.
13. Yoo, C. and Cooper, F. 2003. A computer-based microarray experiment design-system for gene-regulation pathway discovery. *AMIA Annu Symp Proc-2003*: 733-7.
14. Druzdzel, M., and Simon, H. 1993. Causality in Bayesian Belief Networks. In *Proceedings of Ninth Conference on Uncertainty in Artificial Intelligence (UAI-93),* 3-11.
15. Heckerman, D., Geiger, D. and Chickering, D. 1995. Learning Bayesian Networks, in The Combination of Knowledge and Statistical Data. *Machine Learning,* 20(3): 197-243.
16. Kang, C. and Tian, J. 2007. Polynomial Constraints in Causal Bayesian Networks. In *the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)* 200-208.
17. Valtorta, M., and Huang, Y. 2006. Identifiability in Causal Bayesian Networks: A Sound and Complete Algrithm. In *Twenty-First National Conference on Artificial Intelligence (AAAI-06),* 1149-1154.
18. Pearl, J. 2000. *Causality: Models, Reasoning, and Inference.* New York, USA: Cambridge University Press.
19. Tian, J. and Pearl, J. 2002. On the Testable Implications of Causal Models with Hidden Variables. In *Proceedings of the Eighteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-02),* 519-527.
20. Tian, J. 2004. Identifying conditional causal Effects. In *Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence (UAI-04),* 561-568.

21. Tian, J. and Pearl, J. On the identification of causal effects. *Artificial Intelligence,* accepted.
22. Shpitser, I. and Pearl, J. 2006. Identification of joint interventional distributions in recursive semi-markovian causal models. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06),* 1219-1226.
23. Pearl, J.and J.M.R. 1995. Probabilistic evaluation of sequential plans from causal models with hidden variables. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, 444-453.
24. Verma, T. and Pearl, J. 1990. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*, 255-270.
25. Sommerville, L., 2004. *Software Engineering*. 6th ed. New York, NY, USA: Addison Wesley.
26. Feigenbaum, B. 2006. SWT, Swing or AWT: Which is right for you? http://www.ibm.com/developerworks/grid/library/os-swingswt/
27. *Packaging Programs in JAR Files*. http://java.sun.com/docs/books/tutorial/deployment/jar/.
28. Kuroki, M. 1999. Identifiability criteria for causal effects of joint interventions. *Journal of the Japan Statistical Society,* **29(2)**: 105-117.