Aalborg University
Department of Computer Science

Fredrik Bajersvej 7 E • 9220 Aalborg Ø • Denmark

MULTI-RELATIONAL DECISION TREE
BASED ON SELECTION GRAPH

MASTER THESIS

Nguyen Ba Tu
tunb@cs.aau.dk

Jorge Arturo Sánchez Flores
jsanchez@cs.aau.dk

Aalborg, June 2006

Abstract

The area of data mining has been studied for years. The main idea is to try to find useful information from large amounts of data. Many algorithms have been developed for this purpose, but one of the drawbacks from these algorithms is that they only work on single tables. Unfortunately most of the data stored in the real-world are relational databases, consisting of multiple tables and their associations. In order to solve the problem of having relational data, a technique used is multi-relational data mining. This area encompasses multi-relational association rule discovery, multi-relational decision trees and multi-relational distance based methods, among others.

In this thesis we explore the area of multi-relational data mining focusing on multi-relational decision tree based on selection graph. We go from the theoretical introduction to the practical aspects. Firstly, we review the existing concepts of selection graph to show disadvantage points on these. Then, we introduce the formal definition of selection graph. Secondly, we implement the multi-relational decision tree learning algorithm based on selection graph. Finally, we run some experiments and obtain the results from our implementation. We compare them with the results of a commercial software for data mining to estimate the quality of our methodology.

Preface

This thesis was written during the Spring Semester of 2006, as the result of our studies at Aalborg University, to fulfill the requirements for the degree of Master of Science.

Acknowledgements First of all, we would like to thank our supervisor Manfred Jaeger, for his valuable help during the development of this thesis, for his comments and suggestions when writing the thesis.

Ba Tu dedicates this thesis to his parents, to his family and to his girlfriend.

Ba Tu expresses his deep gratitude to the Danish Government for free of tuition fee for his studies.

Jorge dedicates this thesis to his parents Jorge and Angeles, to his brothers Abraham, Isaac and to his sister Gela.

Jorge was supported by the Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, scholarship No.(E04M029937MX).

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we present a brief introduction to knowledge discovery in database and give the motivation of our thesis.

1.1 Knowledge Discovery in Database

Nowadays, the capabilities for collecting data are changing rapidly. Millions of databases are being used in fields such as business management, government administration, scientific and engineering data management and many others. Most of these databases are relational databases. The large amount of data collected and stored might contain some information, which could be useful, but it is not obvious to recognize, nor trivial to obtain it. Sifting through such amounts of data is impossible for humans and even some existing algorithms are inefficient when trying to solve this task. This has generated a need for new techniques and tools that can intelligently and automatically transform the stored data into useful information and knowledge. Data mining is recommended as a solution for this problem. This solution relies heavily on the area of computer science called machine learning, but it is also influenced by statistics, information theory and other fields.

When studying literature about data mining, we have encountered with terms such like: data mining and knowledge discovery in databases (KDD). In various sources [1, 2], those terms are explained on rather different way. A clear definition of data mining is presented in [3]:

> *"Data mining is the process of extracting valid, previously unknown, comprehensible, and actionable information from large databases and using it to make crucial business decisions."*

A different view is presented in [4]:

> *"Knowledge discovery in databases (often called data mining) aims at the discovery of useful information from large collections of data."*

Also through the literature about the topic, the terms data mining and KDD sometimes are used without distinction. In this thesis, we do not go through the comparison of both concepts, but we use them with the same meaning – the discovery of useful information from large collections of data. Basically, the process of knowledge discovery consists of three subtasks [5]:

- Pre-processing of data: this task adapts the original format of the data to fit the input format of the data mining algorithm.
- Data mining: once formatted the data, one or more algorithms must be applied to extract patterns, regularities or general rules from the data.
- Post-processing of result: sometimes the result requires to be turned or translated to more intelligible format.

Almost all data mining algorithms currently available, are based on datasets consisting of a single table. These algorithms only allow the analysis of fairly simple objects. It requires that each object be described by a fixed set of attributes stored in a single table. When we need to represent more complex objects that are stored in a relational database containing multiple tables, each object can be described by multiple records in multiple tables. To be able to analyse relational databases containing multiple relations properly, we need to write specific algorithms to cope with the structural information that occurs in relational databases. The multi-relational data mining framework described in this thesis is one of the solutions.

1.2 Motivation

Data mining algorithms look for patterns in data. While most existing data mining approaches look for patterns in a single data table, multi-relational data mining approaches look for patterns that involve multiple related tables from a relational database. In recent years, multi-relational data mining encompasses multi-relational association rule discovery, multi-relational decision trees and multi-relational distance based methods, among others.

The multi-relational data mining framework proposed in [6] is a novel approach that exploits the semantic information in the database using Structured Query Language (SQL) to learn directly from data in a relational database. Based on this framework, several algorithms for multi-relational data mining have been developed.

- The same authors of [6], in [7], introduce a general description of a decision tree induction algorithm, based on the multi-relational data mining framework and logical decision tree. There are no experimental results available concerning the performance of the algorithm for induction of decision trees from a relational database proposed in [7].
- Based on [5], implementation and experiments reported by [8] have shown that MRDTL - A multi-relational decision tree learning algorithm is competitive with other approaches to learning from relational data.
- Moreover, other implementation and experiments (MRDTL-2) reported by [9] have shown that running time of the implementation in [8] is slow. Therefore, authors turns the algorithm to speed up multi relational decision tree learning.

In this master thesis, we want to implement, experiment and improve the current techniques for multi-relational data mining using selection graphs.

1.3 Organization of the Thesis

Chapter 2 introduces the problem definition. Chapter 3 describes in more detail the multi-relational data mining framework, and discusses our implementation. Chapter 4 presents the results of our experiments. Finally, chapter 5 concludes the thesis and gives possible future directions.

Chapter 2

Problem Definition

In this chapter we review the fundamental concepts in relational databases and we define the problem formulation.

2.1 Relational Database

We will assume that the data to be analyzed is stored in a relational database. A relational database consists of a set of tables and a set of associations (i.e. constraints) between pairs of tables describing how records in one table relate to records in another table. Each table stores a set of objects that have the same properties. Each table consists of rows and columns. Each row of the table represents one record corresponding to one object. Each column is one attribute (property) of an object. For example, we have MOVIE table, the following Table 2.1 shows the table structure.

ID	**Name**	**Producer**	**StudioName**
1	Number thirteen	Hitchcock	Islington
2	Elstree Calling	Brunel	BIP Elstre
3	Lifeboat	MacGowan	Fox

Table 2.1: Illustration of table structure

In the above example, MOVIE table consists of three rows and four columns (ID, Name, Producer, StudioName). Each row is the description of one movie, e.g., the second row is the description of the second movie (2, Elstree Calling, Brunel, BIP Elstre). Each column corresponds to one attribute of the movie, e.g. the second column is the 'name' attribute. All elements in this column store all names of movie. The element at the second row and the second column is 'Elstree Calling', which denotes that the second movie name is 'Elstree Calling'.

In this thesis, we use the notation in Table 2.2

Definition 1 *The domain of the attribute T.A is denoted as DOM(T.A) and is defined as the set of all different values that the records from table T can have in the column of attribute A.*

Definition 2 *A primary key of table T, denoted as T.ID, has a unique value for each row in this table.*

Notation	Meaning
T	Table
D	Database
T.ID	ID is a primary key of T
T.A	A is an attribute of T

Table 2.2: Notation used

Definition 3 *A foreign key in table T_2 referencing table T, denoted as T_2:T.ID, takes values belong to DOM(T.ID).*

An association between two tables describes the relationships between records in both tables. It is defined through primary and foreign keys. The relationship is characterised by the multiplicity of the association. The multiplicity of an association determines whether several records in one table relate to single or multiple records in the second table:

- One-to-one. A record in table T is associated with at most one record in table T_2, and a record in T_2 is associated with at most one record in T.
- One-to-many. A record in table T is associated with any number (zero or more) of records in table T_2. A record in T_2, however, can be associated with at most one record (entity) in T.

Also, the multiplicity determines whether every record in one table needs to have at least one corresponding record in the second table.

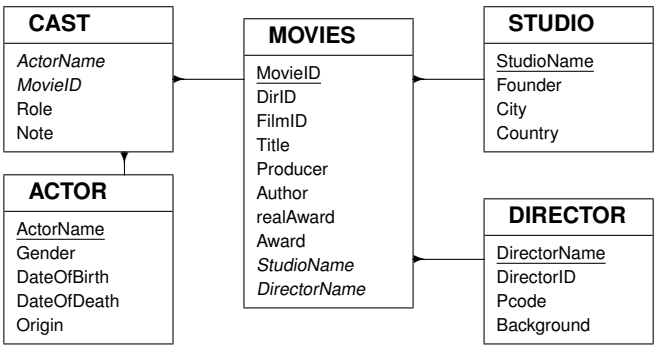
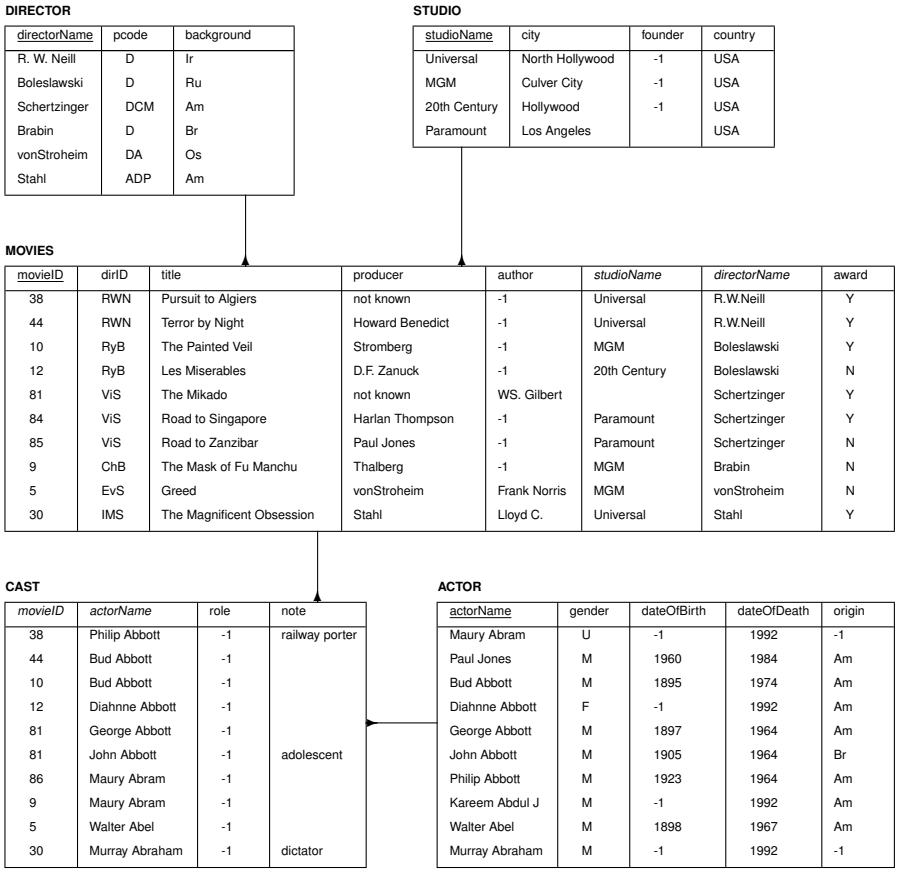


Figure 2.1: MOVIES database schema

Example 2.1:

The data model in Figure 2.1 describes DIRECTOR, MOVIES, STUDIO, CAST and ACTOR as well as how each of these relate to each other. We also use this database schema throughout this thesis. The database schema shows that director may have zero or more movies. Each studio also produces zero or more movies. Cast corresponds to the association between role and actor in each movie. Figure 2.2 shows data used as sample data in this thesis.



DIRECTOR

directorName	pcode	background
R. W. Neill	D	Ir
Boleslawski	D	Ru
Schertzinger	DCM	Am
Brabin	D	Br
vonStroheim	DA	Os
Stahl	ADP	Am

STUDIO

studioName	city	founder	country
Universal	North Hollywood	-1	USA
MGM	Culver City	-1	USA
20th Century	Hollywood	-1	USA
Paramount	Los Angeles		USA

MOVIES

movieID	dirID	title	producer	author	*studioName*	*directorName*	award
38	RWN	Pursuit to Algiers	not known	-1	Universal	R.W.Neill	Y
44	RWN	Terror by Night	Howard Benedict	-1	Universal	R.W.Neill	Y
10	RyB	The Painted Veil	Stromberg	-1	MGM	Boleslawski	Y
12	RyB	Les Miserables	D.F. Zanuck	-1	20th Century	Boleslawski	N
81	ViS	The Mikado	not known	WS. Gilbert		Schertzinger	Y
84	ViS	Road to Singapore	Harlan Thompson	-1	Paramount	Schertzinger	Y
85	ViS	Road to Zanzibar	Paul Jones	-1	Paramount	Schertzinger	N
9	ChB	The Mask of Fu Manchu	Thalberg	-1	MGM	Brabin	N
5	EvS	Greed	vonStroheim	Frank Norris	MGM	vonStroheim	N
30	IMS	The Magnificent Obsession	Stahl	Lloyd C.	Universal	Stahl	Y

CAST

movieID	*actorName*	role	note
38	Philip Abbott	-1	railway porter
44	Bud Abbott	-1	
10	Bud Abbott	-1	
12	Diahnne Abbott	-1	
81	George Abbott	-1	
81	John Abbott	-1	adolescent
86	Maury Abram	-1	
9	Maury Abram	-1	
5	Walter Abel	-1	
30	Murray Abraham	-1	dictator

ACTOR

actorName	gender	dateOfBirth	dateOfDeath	origin
Maury Abram	U	-1	1992	-1
Paul Jones	M	1960	1984	Am
Bud Abbott	M	1895	1974	Am
Diahnne Abbott	F	-1	1992	Am
George Abbott	M	1897	1964	Am
John Abbott	M	1905	1964	Br
Philip Abbott	M	1923	1964	Am
Kareem Abdul J	M	-1	1992	Am
Walter Abel	M	1898	1967	Am
Murray Abraham	M	-1	1992	-1

Figure 2.2: Sample of the MOVIES database

2.2 Objects and Patterns

In a relational database, the data model consists of multiple related tables. One table represents one kind of objects. Each row is one record corresponding to a single object. The purpose of multi-relational data mining will be to predict the objects based on a class label in a relational database. We will refer to descriptions of potentially interesting sets of objects as multi-relational patterns, or simply patterns when clear from the context.

2.3 Problem Formulation

The data model consists of multiple tables, we can choose several kinds of related objects or single kind of objects as central to the analysis. In this approach, we choose single kind of objects we want to analyse, by selecting one of the tables as central (called the target table). Each record in the target table will correspond to a single object in the database. Any information pertaining to the object which is stored in other tables can be looked up by following the associations in the data model. If the data mining algorithm requires a dependent attribute for classification, we can define a particular target attribute within the target table. The purpose of multi-relational data mining is the discovery of useful information from large relational database. At present, there are many

research directions such as multi-relational association rule discovery, multi-relational decision trees and multi-relational distance based methods. We only focus on multi-relational decision tree learning. Therefore, we introduce the problem formulation as follows:

Given:	Data stored in relational database
Goal:	Build multi-relational decision tree for predicting target attribute in target table

Example 2.2:

Given:	Data was stored in MOVIE relational database schema
Goal:	Build multi-relational decision tree for predicting whether movie has received an award or not

Chapter 3

Multi-Relational Data Mining

In this chapter, we introduce the formal definition of selection graph, and then present the multi-relational decision tree based on that concept. We conclude this chapter by describing the system architecture and the implementation details.

3.1 Introduction to Selection Graph

The multi-relational data mining framework was first introduced by Knobbe and colleague in 1999 [6]. The multi-relational data mining framework is based on the search for interesting patterns in the relational database, where patterns can be viewed as subsets of the objects from the database having some properties. In order to describe the selected patterns, they defined the concept of *selection graph*. Basically, selection graphs can be represented graphically as labeled directed graphs and are used to represent selections of objects. The selection graphs also can be transformed directly to SQL query.

Before we introduce the formal selection graph described in section 3.3, we review the existing concepts of selection graph in the following section.

3.2 Existing Selection Graph Concepts

The concept of selection graphs was first introduced by Knobbe and colleague [7]. In order to understand the concept, we will give an overview of the original definition.

Definition 4 *A selection graph S is a directed graph (N, E), where N is a set of triples (T, C, s) called selection nodes, T is a table in the data model and C is a, possibly empty, set of conditions on attributes in T of type T.A operator c; the operator is one of the usual selection operators, =, >, <, etc (for example STUDIO.name = 'MGM'). s is a flag with possible values open and closed. E is a set of tuples (P, Q, a, e) called selection edges, where P and Q are selection nodes and a is an association between P.T and Q.T in the data model (for example* $T_1.ID = T_2{:}T_1.ID$*). e is a flag with possible values present and absent. The selection graph contains at least one node* n_0 *that corresponds to the target table* T_0*.*

Selection graphs can be represented graphically as labeled directed graphs. This means that it can be described as a directed graph with its nodes having a label. The label could be a table name or a table name with a condition of some sort on one of the attributes of the table that the node represents.

Example 3.3:

The following is an example of a selection graph with two nodes:

Figure 3.1: Simple example of a selection graph

In Figure 3.1, the label of the left node is MOVIES corresponding to MOVIES table, and the right one is STUDIO corresponding to STUDIO table.

The nodes in graph have the flag *s*. The value of *s* is indicated by the absence or presence of a cross in the node, representing the value open and closed respectively. When the node's *s* flag is open it means that the selection graph can be extended from that node by including an edge. When the node's *s* flag is closed, there cannot be further growth through that node. In Figure 3.1, the MOVIES node is open, but STUDIO node is closed.

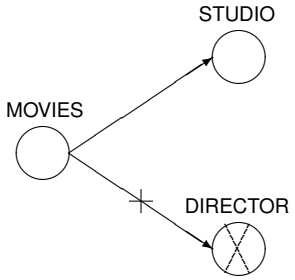


Figure 3.2: Properties of edges in a selection graph

An edge between two nodes can be translated as relation between two tables in a relational dataset. A relation *R* between two tables T_1 and T_2 can be said to be of type T_1.ID = T_2:T_1.ID, where ID is a primary key in table T_1 and ID is also the foreign key in T_2. The edge also has an *e* flag. The value of *e* is indicated by the absence or presence of a cross on the arrow, representing the value present and absent respectively. Present means that there exists *R* described above between two tables. *R* maps tuples in T_1 to T_2 and selects tuples that are equal. On the other hand, absent selects the complement of the tuples selected by present. In Figure 3.2, the edge between MOVIES and STUDIO node is present, the edge between MOVIES and DIRECTOR node is absent.

In Hector's thesis, when extending the selection graph to represent a subset (also called refinement), he introduced one disadvantage. In order to solve this disadvantage case, in [8], Hector L., Anna A. and Vasant H. changed Knobbe's definition of selection graph by adding the *f* flag into selection nodes. *f* is a flag with possible values front and back. It is used to compare how two nodes are connected between an edge. It indicates which node comes before the other in context of each edge. We introduce this selection graph definition as follows:

Definition 5 *A selection graph S is a directed graph, includes a pair (N,E), where N is a set of tuple (T, C, s, f) called selection nodes. T is a table in the data model and C is a, possibly empty, set of conditions on attributes in T of type T.A operator c; the operator is one of the usual selection operators, =, >, <, etc (for example STUDIO.name = 'MGM'). s is a flag with possible values open and closed. f is a flag with possible values front and back.*

E is a set of tuples (P, Q, a, e) called selection edges, where P and Q are selection nodes and a is an association between P.T and Q.T in the data model (for example T_1.ID = T_2:T_1.ID). e is a flag with possible values present and absent.

The selection graph contains at least one node n_0 that corresponds to the target table T_0.

Rule to set value for f flag

Flag f is set to front for those nodes that on their path to n_0 have no closed edges. For all the other nodes flag f is set to back. All refinements can only be applied to the open, front nodes in the selection graph S.

We will explain why they need f flag using the following selection graph. This selection graph includes two branches: top and bottom branch. The top branch is denoted by regular label on nodes. The bottom branch is denoted by the italic label on nodes.

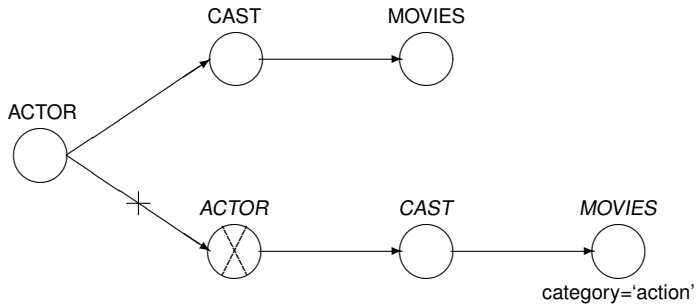


Figure 3.3: Correct 'add negative condition' refinement

If we have no flag f, we can extend CAST or MOVIES node in the bottom branch. Because they are open node. When we can extend, we can get the refinement of above selection graph. This means that the refinement can return greater number of records than the above selection graph does. This is a contradiction with the refinement principle: *the refinement returns less number of records than its selection graph does*.

If we have flag f, with the above rule, all nodes in bottom branch will be set to back value and all nodes in top branch will be set to front value. This means that we only make refinements based on the nodes in the top branch. They still keep the refinement principle.

When consider the f flag, we discover that it is valid if selection graph structure is a tree. Because we have only one path from considering node to the node n_0 corresponding to target table, so we are easy to set value for f flag of that node. The selection graph in Figure 3.4 will show our statement.

With the selection graph in Figure 3.4, we cannot set value for f flag of MOVIES node. Because f flag of MOVIES node is front if we choose the path: MOVIES⟶ CAST⟶ ACTOR. But it is also back if we choose the path: MOVIES⟶ CAST⟶ ROLE⟶ ACTOR.

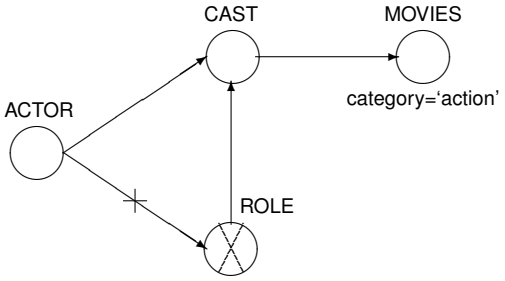


Figure 3.4: Wrong case of f flag

3.3 Formal Selection Graph

3.3.1 Formal Definition

Based on Knobbe's definition and the discovering of Hector A.L., Anna A. and their colleagues, we introduce the different definition of selection graph that we will use in this thesis:

Definition 6 *A selection graph S is a tree (N, E), where N is a set of triples (T, C, k) called selection nodes, T is a table in the data model and C is a, possibly empty, set of conditions* ($C = \{c_1, ..., c_n\}$) *on attributes in T of type T.A operator v; the operator is one of the usual selection operators, =, >, <, etc. and v is any possible value in T.A (for example STUDIO.name = 'MGM'); k is a flag with possible values extend and non-extend. E is a set of tuples (P, Q, e, a) called selection edges, where P and Q are different selection nodes; e is a flag with possible values present and absent; a is an association between P.T and Q.T in the data model, is defined by the relation between the primary key in P and a foreign key in Q, or the other way around (for example* $T_1.ID = T_2{:}T_1.ID$*). The selection graph contains at least one root node* n_0 *that corresponds to the target table* T_0.

In this definition, instead of using the s and f flags as in [7], [8], [9] and [10], we use the k flag. The value of k is indicated by the absence or presence of a cross in the node, representing the value extend and non-extend respectively. When the node's k flag is extend it means that the selection graph can be extended from that node by including an edge or condition. When the node's k flag is non-extend, there cannot be further growth through that node. For example, in Figure 3.5, the MOVIES node is extend, but STUDIO node is non-extend.

Figure 3.5: Selection graph with two nodes

3.3.2 Semantic of Selection Graph

Selection graphs represent selections of patterns in relational database. The selection node N represents a selection of records in the corresponding table $N.T$ which is determined by the set of conditions $N.C$ and the relationship with records in other tables characterised by selection edges con-

nected to N. In other words selection graphs represent subset of objects from relational database having some properties.

For the purpose of this section we will refer to the tuple relational calculus [11], expressed as

$$\{t \mid P(t)\}$$

it is the set of tuples t such that predicate P is true for t. We use $t[A]$ to denote the value of tuple t on attribute A, and $t \in T$ to denote that tuple t is in table T.

Example 3.4:

The simplest selection graph consists of one node corresponding to target table, as the following Figure shows.

Figure 3.6: The simplest selection graph

The graph in Figure 3.6 represents all records in MOVIES table.

$$\{t \mid t \in MOVIES\} \tag{3.1}$$

movieID	dirID	title	producer	author	studioName	directorName	award
38	RWN	Pursuit to Algiers	not known	-1	Universal	R.W.Neill	Y
44	RWN	Terror by Night	Howard Benedict	-1	Universal	R.W.Neill	Y
10	RyB	The Painted Veil	Stromberg	-1	MGM	Boleslawski	Y
12	RyB	Les Miserables	D.F. Zanuck	-1	20th Century	Boleslawski	N
81	ViS	The Mikado	not known	WS.Gilbert		Schertzinger	Y
84	Vis	Road to Singapore	Harlan Thompson	-1	Paramount	Schertzinger	Y
86	Vis	Road to Zanzibar	Paul Jones	-1	Paramount	Schertzinger	N
9	ChB	The Mask of Fu Manchu	Thalberg	-1	MGM	Brabin	N
5	EvS	Greed	vonStroheim	Frank Norris	MGM	vonStroheim	N
30	IMS	The Magnificent Obsession	Stahl	Lloyd C.	Universal	Stahl	Y

Table 3.1: Semantic of the simplest selection graph

Example 3.5:

Now we have a more complex selection graph, composed of two nodes and one condition.

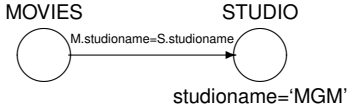


Figure 3.7: Simple selection graph

The graph in Figure 3.7 represents the records in MOVIES table that have studioname equals to 'MGM'.

$$\{t \mid \exists\, s \in STUDIO(t[studioname] = s[studioname] \wedge s[studioname] = `MGM')\} \tag{3.2}$$

Figure 3.2 shows the records in the database that are represented by selection graph in Figure 3.7.

movieID	dirID	title	producer	author	studioName	directorName	award
10	RyB	The Painted Vell	Stromberg	-1	MGM	Boleslawski	Y
9	ChB	The Mask of Fu Manchu	Thalberg	-1	MGM	Brabin	N
5	EvS	Greed	vonStroheim	Frank Norris	MGM	vonStroheim	N

Table 3.2: Semantic of the simple selection graph

Decomposition of Selection Graph

We now have a more complex selection graph, we can decompose it to many simpler selection graphs. The simpler selection graphs will still keep similar semantics. Figure 3.8.a is the complex selection graph, b and c are the simple one that we decompose from a.

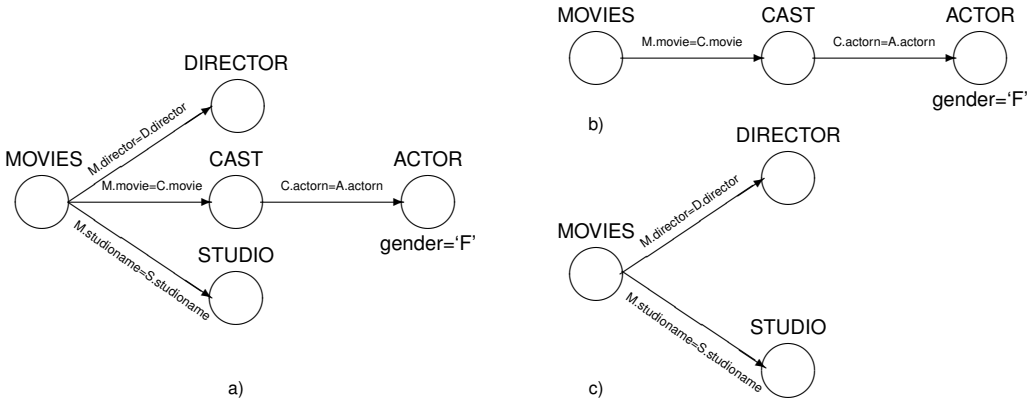


Figure 3.8: Decomposition of selection graph

The selection graph in Figure 3.8.a returns the subset of object (the movies that have at least one female actor and have both studioname and directorname).

movieID	dirID	title	producer	author	studioName	directorName	award
12	RyB	Les Miserables	D.F.Zanuck	-1	20th Century	Boleslawski	N

Table 3.3: Semantic of the selection graph in Figure 3.8.a

The selection graph in Figure 3.8.b returns the subset of object (the movies that their actor's gender is equal to 'F').

$$\begin{aligned}\{t \mid \exists\, s \in CAST(t[movieID] = s[movieID] \\ \wedge \exists\, u \in ACTOR(s[actorName] = u[actorName] \wedge u[gender] = F))\}\end{aligned} \tag{3.3}$$

The selection graph in Figure 3.8.c returns the subset of object (the movies have both studioname and directorname).

movieID	dirID	title	producer	author	studioName	directorName	award
12	RyB	Les Miserables	D.F.Zanuck	-1	20th Century	Boleslawski	N

Table 3.4: Semantic of the selection graph in Figure 3.8.b

$$\{t \mid \exists\, s \in DIRECTOR(t[directorName] = s[directorName] \\ \wedge \exists\, u \in STUDIO(s[studioName] = u[studioName]))\} \tag{3.4}$$

movieID	dirID	title	producer	author	studioName	directorName	award
38	RWN	Pursuit to Algiers	not known	-1	Universal	R.W.Neill	Y
44	RWN	Terror by Night	Howard Benedict	-1	Universal	R.W.Neill	Y
10	RyB	The Painted Veil	Stromberg	-1	MGM	Boleslawski	Y
12	RyB	Les Miserables	D.F.Zanuck	-1	20th Century	Boleslawski	N
84	Vis	Road to Singapore	Harlan Thompson	-1	Paramount	Schertzinger	Y
86	Vis	Road to Zanzibar	Paul Jones	-1	Paramount	Schertzinger	N
9	ChB	The Mask of Fu Manchu	Thalberg	-1	MGM	Brabin	N
5	EvS	Greed	vonStroheim	Frank Norris	MGM	vonStroheim	N
30	IMS	The Magnificent Obsession	Stahl	Lloyd C.	Universal	Stahl	Y

Table 3.5: Semantic of the selection graph in Figure 3.8.c

We define that a subset of objects in a complex selection graph is equal to the intersection of all objects in its subgraphs. Therefore, the subset of objects in the complex selection graph is often less than or equals to union of all subset in its subgraphs.

From the several existing selection graph, also we can compose them into one more complex selection graph using the intersection of all objects in the subgraphs.

After the presented example, we can now give the formal semantic for selection graphs using the following inductive definition.

SEMANTIC_DEFINITION(S: Selection Graph)

Begin

1. S has no branch
$$Sem(S) = \{t|\ t \in T(C_j),\ j = 1, ..., m\} \tag{3.5}$$
where $T(C_j)$, are the conditions on a table and define a subset of the table.

2. S has n branches ($n \geq 1$)

   Let us assume that we have both semantics of $n-1$ branch selection graph and *new* added one: $Sem(S_{n-1})$ and $Sem(S_{new})$, respectively (Figure 3.9).

   if the new added edge is open then
$$Sem(S_n) = \{t_n \in T|\ t_n \in Sem(S_{n-1}) \wedge \exists u(u \in Sem(S_{new}) \wedge u[A] = t_n[B]\} \tag{3.6}$$
   where $u[A] = t[B]$, is an association between tuples u, t on attribute $[A]$ and, that for each $t \in P$ there exists a set of $u \in Q$.

   else
$$Sem(S_n) = \{t_n \in T|t_n \in Sem(S_{n-1}) \wedge \neg\exists u(u \in Sem(S_{new})\} \wedge u[A] = t_n[B]\} \tag{3.7}$$
   end if

end

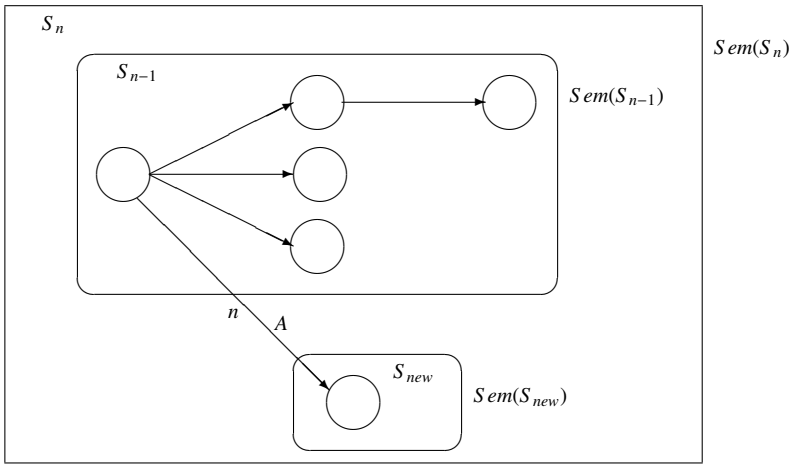


Figure 3.9: Semantic of selection graph

3.3.3 Transformation of Selection Graph to SQL Statement

We need a new technique (tools) that can manipulate the data in relational database efficiently. The shortest way to manipulate data in relational database is with SQL statements. As explained above, selection graphs represent subset of objects from relational database having some properties. We introduce the induction algorithm that transforms selection graph into SQL statement to get patterns. The pseudocode is shown in Algorithm 1

This algorithm uses GET_SUBGRAPH(Q) function to return the selection subgraph with root node be Q.

The algorithm effectively produces a join of all the tables associated with an open node. For each subgraph attached to an absent edge, a sub-query is produced by calling the GET_SUBGRAPH procedure. The fact that we state select distinct in the main query rather than select is caused by the fact that a record in target table T_0 may be covered by the selection graph in multiple ways. Since the target table represents our objects, they should be counted only once.

The generic resulting SQL-statement is shown in the following Table 3.6

GENERIC QUERY:
select distinct $T_0.primary\_key$ **from** *table_list* **where** *join_list* and *condition_list* * T_0 **=** *target_table*

Table 3.6: Selection graph translation to SQL-statement

Example 3.6:

We assume that we have the selection graph shown in Figure 3.10. We run Algorithm 2 step-by-step to illustrate how it works.

Algorithm 1 GET_SQL (S)

```
1:  Input: Selection graph S
2:  Output: SQL statement Sql
3:  Init:
        Table_list:=";
        Join_list:=";
        Condition_list:=";
4:  If nodes = 0 then exit;
5:  for each nodes[i] of S do
6:     Table_list.add(nodes[i].T);
7:     Condition_list.add(nodes[i].C);
8:     for j=0 to outgoing edges from nodes[i] do
9:        if edges[j].e = present then
10:          join := 'edges[j].P.ID = edges[j].Q:P.ID';
11:          Join_list.add(join);
12:       else
13:          Sub:= GET_SUBGRAPH(edges[j].Q)
14:          join:= node.ID not in GET_SQL(Sub)
15:          Join_list.add(join);
16:       end if
17:    end for
18: end for
19: return Sql = SELECT root_table.ID
                 FROM table_list
                 WHERE join_list AND condition_list
```

Algorithm 2 GET_SUBGRAPH (root node)

```
1:  Input: root node N
2:  Output: Selection graph S
3:  Init:
        Nodes = ";
        Edges = ";
4:  Nodes = Nodes.add(N);
5:  If edges outgoing from N = 0 then exit;
6:  for each edge outgoing from N do
7:     Edges.add(edge);
8:     GET_SUBGRAPH(edge.Q)
9:  end for
10: return S(Nodes, Edges)
```

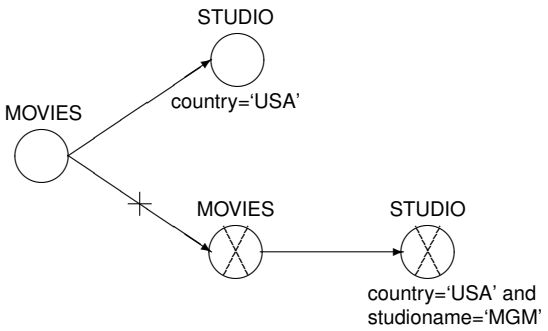


Figure 3.10: Example of a selection graph

Step 1: Initialize the value of variables: *table_list*, *join_list* and *condition_list* to empty.

We start running the algorithm with input parameters(whole selection graph S). First, we have root node. This is MOVIES node. We get value of variables as follows:

- Table_list = MOVIES.
- Condition_list=''.

We have two outgoing edges from MOVIES node:

- With *present* edge MOVIES→STUDIO, we get join_list = $MOVIES.studioname = STUDIO.studioname$.
- With *absent* edge MOVIES→MOVIES, we call GET_SUBGRAPH starting with $non - extend$ MOVIES node. And then, we remove all subgraph starting with $non - extend$ MOVIES node. We can get join_list to add join variable as follows:

```
join = MOVIES.movieID not in
        ( SELECT MOVIES.movieID
          FROM MOVIES, STUDIO
          WHERE MOVIES.studioname=STUDIO.studioname
              AND STUDIO.studioname = MGM
              AND STUDIO.country=USA)
```

Step 2:

We have only one node. This is STUDIO node. We get value of variables as follows:

- Table_list = MOVIES, STUDIO
- Condition_list='STUDIO.studioname=MGM'

We have no edges. Therefore, this algorithm will stop here. Finally, this algorithm returns SQL-statement as follows:

```
SELECT          MOVIES.movieID
FROM            MOVIES, STUDIO
WHERE           MOVIES.studioname = STUDIO.studioname
AND             STUDIO.studioname = MGM
AND             MOVIES.movieID NOT IN (
        SELECT  MOVIES.movieID
        FROM    MOVIES, STUDIO
        WHERE   MOVIES.studioname = STUDIO.studioname
        AND     STUDIO.studioname = MGM
        AND     STUDIO.country = USA)
```

3.3.4 Refinement of Selection Graph

Multi-relational data mining algorithms search for and successively refine interesting patterns. During the search, patterns will be considered and those that are promising will be refined. The idea of search for patterns is basically a top-down search [7]. In order to search, they first introduced the concept of *refinement* and its *complement* in [7]. Basically, a refinement is a selection graph that bests represents the data, based on the prediction we are trying to find. Refinements can be made to find a hypothesis that is consistent with the data. After each refinement, we can reduce the search space and efficiently evaluate potentially interesting patterns [7]. In other words, the refinement principle is *the refinement returns less number of patterns than its selection graph does*. Given the selection graph, there are two methods for refinement: *addition of conditions* and *addition of extended nodes and present edges* and two methods for complement of refinement: *condition complement* and *edge complement*. Below, we will explain in detail how each refinement works using examples from MOVIE database. We will illustrate all refinements on selection graph based on Figure 3.11. The complement of refinement is introduced in section 3.3.5.

Note that a refinement can only be applied to *extended* node in a selection graph.

Figure 3.11: Considering selection graph

We introduce the graph in Figure 3.12, that will be used in this section to show the different examples.

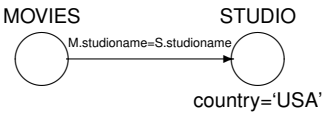


Figure 3.12: Simple selection graph

Add positive condition This refinement adds a condition c to the set of conditions C in the refined node. This refinement does not change the structure in the selection graph S, nor the value of k changes.

Figure 3.13: 'Add positive condition' refinement

Using the selection graph in Figure 3.12 as example, we make the refinement STUDIO. studioname='MGM' as the condition c. This condition c will be added to the set of conditions C for the node STUDIO, in which the condition STUDIO.country='USA' was previously defined. The selection graph in Figure 3.14 shows this refinement. Table 3.7 represents the semantic of the selection graph in Figure 3.12 with one condition. After adding the condition in this example, Table 3.8 represents the semantic for this refinement.

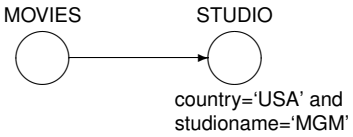


Figure 3.14: 'Add positive condition' refinement example

movieID	title	producer	studioName	directorName	award
154	Pursuit to Algiers	not known	MGM	R.W.Neill	Y
252	Terror by Night	Howard Benedict	MGM		Y
305	The Painted Veil	Stromberg	MGM		Y
367	Les Miserables	D. Zanuck	20th Century	Boleslawski	N
493	The Mikado	not known	Miramax	Schertzinger	Y
508	Road to Singapore	Harlan Thompson	Paramount		Y
529	Zanzibar	Paul Jones	Dreamworks	Paul Jones	N
690	The Mask of Fu Manchu	Thalberg	Fox	Brabin	N
747	Greed	VonStroheim	MGM	VonStroheim	N
760	The Crowd	K.Vidor	MGM	K. Vidor	Y
813	6000 Enemies	Lucien Hubbs	MGM		N
880	The Magnificent Obsession	Stahl	MGM		Y
903	London After Midnight	Stahl	Universal	Franklin	Y
956	Seven Women	Bernard Smith	MGM		N
987	Mogambo	Zimbalist	MGM		Y

Table 3.7: Semantic of the selection graph in Figure 3.12

Add present edge and extended node This refinement adds a present edge and its table as an extended node to the selection graph S.

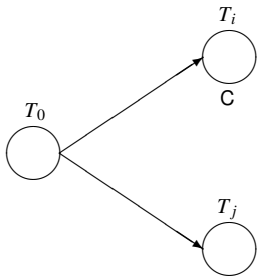


Figure 3.15: 'Add present edge and extended node' refinement

Using the selection graph in Figure 3.12, a new edge from MOVIE node to DIRECTOR node results in the selection graph in Figure 3.16. Table 3.7 is the semantic of the selection graph in Figure 3.12. The semantic for this example is represented in Table 3.9.

movieID	title	producer	studioName	directorName	award
154	Pursuit to Algiers	not known	MGM	R.W.Neill	Y
252	Terror by Night	Howard Benedict	MGM		Y
305	The Painted Veil	Stromberg	MGM		Y
747	Greed	VonStroheim	MGM	VonStroheim	N
760	The Crowd	K.Vidor	MGM	K. Vidor	Y
813	6000 Enemies	Lucien Hubbs	MGM		N
880	The Magnificent Obsession	Stahl	MGM		Y
956	Seven Women	Bernard Smith	MGM		N
987	Mogambo	Zimbalist	MGM		Y

Table 3.8: Semantic of the 'add positive condition' refinement example

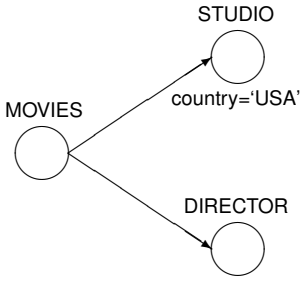


Figure 3.16: 'Add present edge and extended node' refinement example

movieID	title	producer	studioName	directorName	award
154	Pursuit to Algiers	not known	MGM	R.W.Neill	Y
367	Les Miserables	D. Zanuck	20th Century	Boleslawski	N
493	The Mikado	not known	Miramax	Schertzinger	Y
529	Zanzibar	Paul Jones	Dreamworks	Paul Jones	N
690	The Mask of Fu Manchu	Thalberg	Fox	Brabin	N
747	Greed	VonStroheim	MGM	VonStroheim	N
760	The Crowd	K.Vidor	MGM	K. Vidor	Y
903	London After Midnight	Stahl	Universal	Franklin	Y

Table 3.9: Semantic of the 'add present edge and extended node' refinement example

It is worth mentioning that the 'add condition' refinement takes place only on the attributes of the involved table. The exploration of the tables in the database is performed with the 'add edge present edge and extended node' refinement.

Avoiding non-meaningful conditions

When we build a refinement by adding conditions there could be some unexpected selection graphs to occur. For example, the condition *sex = 'male'* is meaningful, but the *sex > 'male'* is a non-meaningful condition. Since we can only compare that condition with two values. Figure 3.17 represents this selection graph.

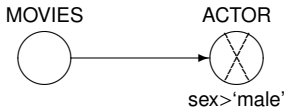


Figure 3.17: Unexpected selection graph

In order to avoid non-meaningful conditions, we use the data type of the column. First, we check the data type of column, and then we make the decision on which operator can be chosen based on this selection. Algorithm 3 shows this pseudocode.

Algorithm 3 CHOOSE_OPERATOR (T, A)

```
1: Input: Table T, Column name A.
2: Output: Operator.
3: SELECT datatype of A INTO type FROM T
4: switch type
5: case: boolean
6:       Operator:= {=}
7: case: numeric
8:       Operator:= one of {<, >, =, <=, >=, <>}
9: case: string
10:      Operator:= one of {=, <>}
11: end switch
12: return Operator
```

3.3.5 Complement of Refinement

In SET theory, the complement concept is defined as follows: *If A and B are sets, then the complement of A in B is the set of elements in B but not in A*. Besides, the semantic of selection graph is the set of interesting objects. Therefore, we introduce the complement concept for selection graph as follows.

Definition 7 *Given a selection graph S and its refinement R(S), the complement of R(S) is a selection graph R_com(S) that contains the elements in S which are not in R(S).*

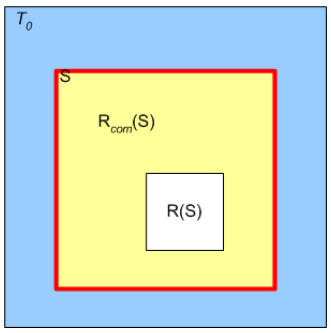


Figure 3.18: Complement of selection graph

In Figure 3.18, the middle bounded square represents all elements of selection graph S. The small inner square represents all elements of refinement selection graph $R(S)$. And the area in between represents all elements of complement of refinement of selection graph $R\_com(S)$.

Now, we introduce the two methods for creating the complement of the refinements described in section 3.3.4.

Condition complement This is used when we create the complement of 'Add positive condition' refinement. When the considering node is the root node n_0, the new condition is negated and added to the list of conditions C in n_0. When the considering node is not the root node n_0, the

complement adds a *new absent edge* from n_0 (corresponding to target table T_0) to a subgraph representing the original graph plus the non-negated condition to the corresponding node's condition list. The value of k flag in all nodes that belong to the subgraph will be set to non-extend.

Remark: when the considering node is not n_0, we do not use the association between n_0 and the node corresponding to T_0 in the subgraph because the two nodes represent T_0. In other words, the association T_0.ID = T_0.ID is redundant.

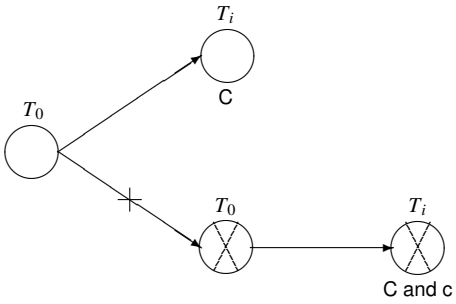


Figure 3.19: Condition complement

Using the selection graph in Figure 3.7, we use the condition 'country=USA' in the condition list, we add a new absent edge from the MOVIES node to the subgraph of MOVIES and STUDIO node, then we add the new condition 'studioname=MGM' into STUDIO node of the subgraph. The k flags in all nodes of the subgraph are set to non-extend. Table 3.10 shows the semantic for this complement, the rows represented by Table 3.10 and Table 3.8 partitions Table 3.7, ensuring that all elements in a selection graph are covered by its refinement and complement.

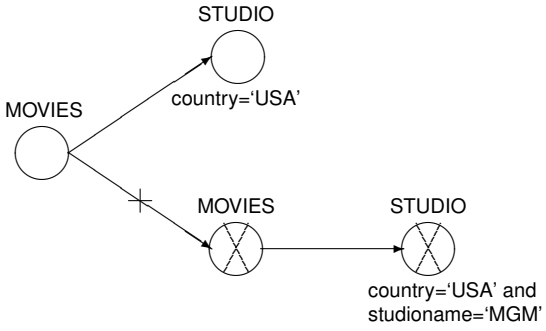


Figure 3.20: 'Condition complement' example

Remark: In Figure 3.20, we do not use the association MOVIES.movieID = MOVIES.movieID between the root node of the selection graph and MOVIES node of the subgraph (MOVIES ⟶ STUDIO).

movieID	title	producer	studioName	directorName	award
367	Les Miserables	D. Zanuck	20th Century	Boleslawski	N
493	The Mikado	not known	Miramax	Schertzinger	Y
508	Road to Singapore	Harlan Thompson	Paramount		Y
529	Zanzibar	Paul Jones	Dreamworks	Paul Jones	N
690	The Mask of Fu Manchu	Thalberg	Fox	Brabin	N
903	London After Midnight	Stahl	Universal	Franklin	Y

Table 3.10: Semantic of the 'condition complement' example

Proposition 1 *The condition complement is the complement of the add positive condition refinement.*

Edge complement This is used when we create the complement of 'Add present edge and extended node' refinement. When the node to be complemented is directly associated to the root node in the selection graph S, we add an absent edge and its corresponding table as a non-extended node (Figure 3.21.a). When the node to be complemented is not directly associated to the root node in the selection graph S, we apply a procedure similar to the 'condition complement' (Figure 3.21.b).

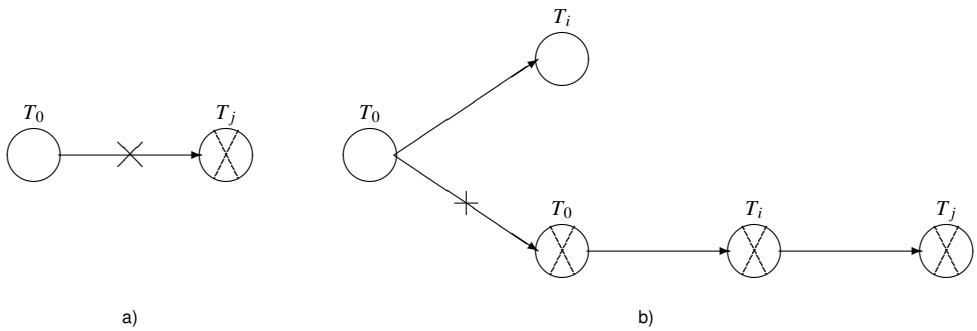


Figure 3.21: Edge complement

Using the selection graph in Figure 3.12, a new absent edge from MOVIE node to DIRECTOR non-extended node results in selection graph (Figure 3.22). Since this is the complement of the 'add present edge and extended node' refinement, Table 3.11 is the semantic for this example.

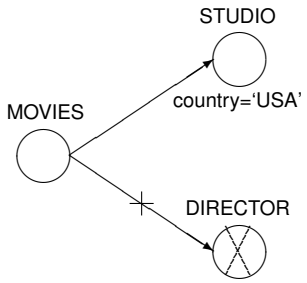


Figure 3.22: 'Edge complement' example

movieID	title	producer	studioName	directorName	award
252	Terror by Night	Howard Benedict	MGM		Y
305	The Painted Veil	Stromberg	MGM		Y
508	Road to Singapore	Harlan Thompson	Paramount		Y
813	6000 Enemies	Lucien Hubbs	MGM		N
880	The Magnificent Obsession	Stahl	MGM		Y
956	Seven Women	Bernard Smith	MGM		N
987	Mogambo	Zimbalist	MGM		Y

Table 3.11: Semantic of the 'edge complement' example

Proposition 2 *The edge complement is the complement of the add present edge and extended node refinement.*

3.3.6 Exploring the Refinement Space

The exploration of conditions represents time spent in building the tree, as the number of conditions increases the time taken increases as well. The number of 'add positive condition' refinements can become quite big, depending on the number of the distinct values in each column. Because the conditions were constructed by format as *column_name - operator - value*. Besides, we known that some conditions are only present in a few instances in the table of the database, while others are present in a lot of instances. Therefore, it would be a waste of time trying the conditions that presents a few instances. To avoid the exploration of all the conditions, we decided to try only a few conditions that present a lot of instances.

In order to do this, we create an array of conditions involving the condition and the count of instances corresponding to each condition. The conditions are listed in descending order of count and only a few selection of conditions from the top are tested by the 'add positive condition' refinement. In other words, the requirement for a condition to be tested is that it has to be in several instances in the table of the database. The Algorithm 4 is the pseudocode to get the most frequent value to create the condition for the considering column.

Algorithm 4 GET_CONDITION (T, A)

1: *Input: Table T, Column A.*
2: *Output: Condition.*
3: Get the *value* in $T.A$ that is the most frequence;
4: opt := CHOOSE_OPERATOR(T, A);
5: **return** $T.A$ opt *value*;

3.4 Multi-Relational Decision Tree Learning Algorithm

A tree data structure accessed beginning at the *root* node. Each node is either a *leaf* or an *internal* node. An internal node has one or more *child* nodes and is called the *parent* of its child nodes. Contrary to a physical tree, the root is usually depicted at the top of the structure, and the leaves are depicted at the bottom.

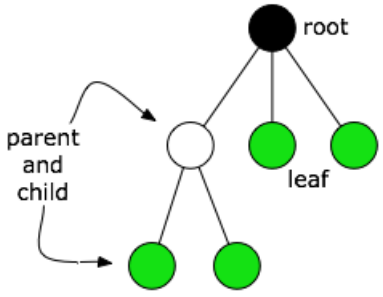


Figure 3.23: Tree data structure

Based on this concept, we introduce the definition of multi-relational decision tree based on selection graph. Basically, a multi-relational decision tree has a data structure similar as a tree, but each node refers to a selection graph.

3.4.1 Multi-Relational Decision Tree Definition

The definition of multi-relational decision tree is as follows:

Definition 8 *A multi-relational decision tree $\mathcal{T}$ is a binary tree ($\mathcal{N}$), where $\mathcal{N}$ is a set of tuples (V, L, C, RC, LC) called tree nodes, V refers to the corresponding selection graph; L is a flag with possible values true or false, represented whether node is leaf or not; C is a class label, empty for non-leaf nodes; RC and LC represent the identification of the right and left child nodes, respectively. And $\mathcal{T}$ has the following properties:*

The left child node refers to the refinement of the selection graph of the parent node.

The right child node is the complement of the left child node.

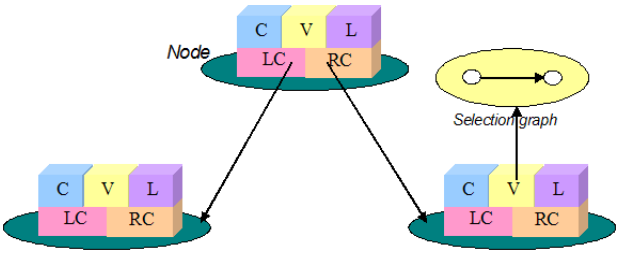


Figure 3.24: Structure of multi-relational decision tree

3.4.2 Multi-Relational Decision Tree Construction

In [6], Knobbe and colleagues introduced an algorithm for top-down induction of multi-relational decision tree within the multi-relational data mining framework. It illustrated the use of selection graphs, and specifically the use of complementary selection graphs in the second branch of a split. In order to search interesting patterns, where patterns can be viewed as subsets of the objects from the database having some properties. The most interesting subsets are chosen according to some measures (i.e. information gain for classification task), which guides the search in the space of all patterns. The search for interesting patterns usually proceeds by a top-down induction. For each interesting pattern, sub-patterns are obtained with the help of refinement operator, which can be seen as further division of the set of objects covered by initial pattern. Top-down induction of interesting pattern proceeds recursively applying such refinement operators to the best patterns. The pseudocode is shown in Algorithm 5.

In order to initiate the algorithm for constructing the tree, we need two input parameters. The first is the selection graph. This selection graph S will be the initial node and it will represent the target table T_0, with the attribute of interest. The second is the relational database D, this will be the hypothesis space where the algorithm will search to discover patterns.

The algorithm starts with a selection graph including a single node at the root of the tree which represents the target table T_0. By analyzing all possible refinements of the considering selection graph, and examining their quality by applying some measures (e.g. information gain), we determine the optimal refinement. This optimal refinement, together with its complement, are used to create the patterns associated with the left and the right branch respectively. Based on the stopping criterion it may turn out that the optimal refinement and its complement do not give cause for

Algorithm 5 BUILD_TREE (S: selection graph, D: database)

```
1: Input: Selection graph S, Database D
2: Output: Root of multi-relational decision tree T.R
3:
4: Create root node of tree T.R;
5: if stopping_criterion(S) then
6:    return T.R;
7: else
8:    R:=optimal_refinement(S);
9:    T.R.LC := BUILD_TREE(D, R(S));
10:   T.R.RC := BUILD_TREE(D, R_com(S));
11: end if
12: return T.R;
```

further splitting, a leaf node is introduced instead. Whenever the optimal refinement does provide a good split, a left and right branch are introduced and the procedure is applied to each of these recursively.

Example 3.7:

In order to illustrate how Algorithm 5 works, we apply it to a classification problem within MOVIE example database. The problem is described in section 2.3. We are still predicting whether movies get an award or not.

We start with an initial selection graph which represents the target table MOVIE corresponding to all movies in our database (Figure 3.25).

Figure 3.25: Initial selection graph

By running *optimal_refinement* function (detailed description in section 3.4.4) using 'add condition' and 'add edge' refinement, we assume that we can get the following list of possible refinements:

- Add positive condition MOVIES.producer = 'Hitchcock'.
- Add positive condition MOVIES.dirID = 'H'.
- Add present edge and extended node from MOVIES to STUDIO.
- Add present edge and extended node from MOVIES to DIRECTOR.

In *optimal_refinement* function, every refinement is tested resulting that STUDIO produces many MOVIES is the best choice. The following selection graph and its complement are created for left and right branch respectively (Figure 3.26).

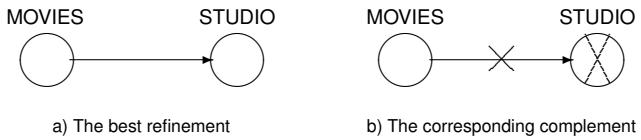


Figure 3.26: Refinement and its complement after one iteration

We check the best refinement in *stopping_criterion* function (described in section 3.4.5). We assume that it does not meet the stopping condition, so the induction process is continued recursively for the set of movies that were produced by studios. At this point in the tree, we assume that we only demonstrate the effect for left branch, and get the following list of refinements, besides the previous list of refinements:

- Add positive condition STUDIO.studioname = 'MGM'.
- Add positive condition STUDIO.country = 'USA'.
- Add present edge and extended node from MOVIES to DIRECTOR.

The same process of finding the optimal refinement is repeated, and this time a condition on the studioname is introduced. The left branch will now represent the set of movies that were produced by MGM studio. The right branch will be the set of movies that were not produced by MGM studio. Figure 3.27 shows the refinement of selection graph and its complement.

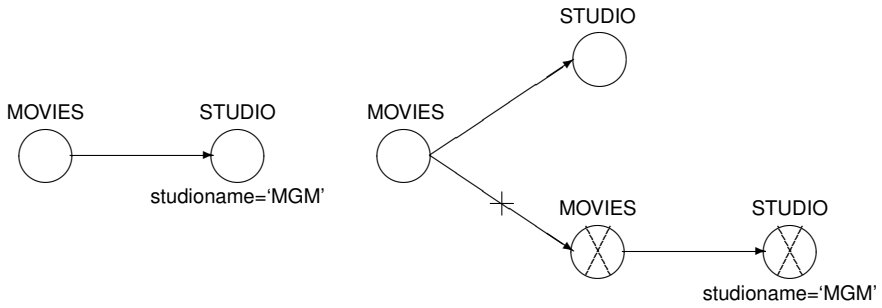


Figure 3.27: Refinement and its complement after two iterations

We check again the best refinement in *stopping_criterion* function. We assume that it meets the stopping condition, so the induction process is stopped. The resulting decision tree will be in the following Figure 3.28.

3.4.3 Partition of Leaf Nodes

Based on Definition 8, each node of multi-relational decision tree represents the corresponding selection graph. Each node of multi-relational decision tree has maximum two children nodes. Selection graphs in left children nodes is refinement of the one in parent node. According to Algorithm 5, at lines 9 and 10, in two children nodes, the selection graph in the right node is the complement of the one in the left node. Therefore, subset in the parent node was always split into two non-overlap subsets. In other words, the subset in root node was split into non-overlap subsets in leaf nodes. Consequently, all leaf nodes partitions the subset of target table into non-overlap subsets. Hence, we state the following proposition.

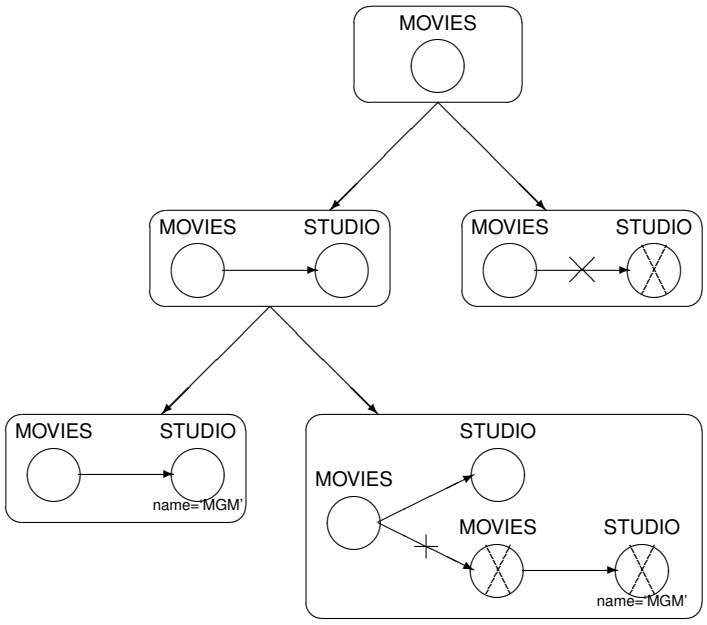


Figure 3.28: Resulting tree

Proposition 3 *The subsets of the target table defined by the leaf node form a partition of the target table.*

3.4.4 Information Gain Associated with Refinements

In Algorithm 5, the function $optimal\_refinement(S)$ considers every possible refinement that can be made to the current selection graph S and selects the (locally) optimal refinement (i.e., one that maximizes information gain). Here, $R(S)$ denotes the selection graph resulting from applying the refinement R to the selection graph S. $R_{com}(S)$ denotes the application of the complement of R to the selection graph S. In order to measure information gain for each possible refinement, some kind of statistic gathering from the database is necessary. Therefore, multi-relational decision tree learning (MRDTL) uses SQL operations to obtain the *counts* needed for calculating information gain associated with the refinements. For that purpose, a series of queries have been proposed in [7], and they are outlined below.

Computing of Counts

Firstly we show the calculation of the information gain with current selection graph S. Let T_0 be the target table. We have the SQL query that returns the counts of current selection graph $count(S)$ as follows:

```
SELECT  T0.target_attribute, COUNT (T0.ID)
FROM    table_list
WHERE   join_list
AND     condition_list
```

Note that only *join_list* and *condition_list* can be empty, but at least one table will be in *table_list*.

Secondly, we show the calculation of the information gain associated with 'add condition' refinement. Let T_i be the table associated with one of the nodes in the current selection graph S and $T_i.A_j$ be the attribute to be refined, and $R_{vj}(S)$ and $R_{com-vj}(S)$ be the add condition $T_i.A_j = v_j$ refinement and its complement respectively. The goal is to calculate entropies associated with the split based on the 'add condition' refinement and 'condition' complement. This requires the following counts: $count(R_{vj}(S))$ and $count(R_{com-vj}(S))$. The result of the SQL query shown below returns a list of the necessary counts: $count(R_{vj}(S))$.

SELECT	T_0.target_attribute, $T_i.A_j$, COUNT (T_0.ID)
FROM	table_list
WHERE	join_list
AND	condition_list and $T_i.A_j = v_j$
GROUP BY	T_0.target_attribute, $T_i.A_j$

The sum of the resulting counts must be equal to the result of prior query that measures the support of a pattern. The rest of the counts needed for the computation of the information gain can be obtained from the following formula:

$$count(R_{com-vj}(S)) = count(S) - count(R_{vj}(S)) \tag{3.8}$$

Finally, consider the SQL query for the calculation of the information gain associated with 'add present edge and extended node' refinement. Let T_i be the table associated with one of the nodes in the current selection graph S and e be the edge to be added from table T_i to table T_j . $R_e(S)$ and $R_{com-e}(S)$ be the add edge e refinement and its complement respectively. The goal is to calculate entropies associated with the split based on the refinement and its complement. This requires the following counts: $count(R_e(S))$ and $count(R_{com-e}(S))$. The result of the SQL query shown below returns a list of the necessary counts: $count(R_e(S))$.

SELECT	T_0.target_attribute, COUNT (T_0.ID)
FROM	table_list
WHERE	join_list AND e
AND	condition_list
GROUP BY	T_0.target_attribute

The sum of these counts can exceed the support of the given pattern if the nominal attribute is not in the target table and multiple records with different values for selected attribute may correspond to a single record in the target table. The rest of the counts needed for the computation of the information gain can be obtained from the following formula:

$$count(R_{com-e}(S)) = count(S) - count(R_e(S)) \tag{3.9}$$

Entropy

Based on the counts that were computed as above, we can calculate entropy of S, $R(S)$, and $R_{com}(S)$ from the following formula:

$$sc = \sum_{c_i \in counts} c_i \tag{3.10}$$

$$p_i = \frac{c_i}{sc} \tag{3.11}$$

$$Entropy(S) = -\sum_{c_i \in counts} p_i * log_2(p_i) \tag{3.12}$$

Information Gain

Based on the entropy of S, $R(S)$, and $R_{com}(S)$, we can calculate the information gain for refinement as follows:

$$Gain = Entropy(S) - \frac{|R(S)|}{|S|} Entropy(R(S)) - \frac{|R_{com}(S)|}{|S|} Entropy(R_{com}(S)) \tag{3.13}$$

3.4.5 Stopping Criterion

The function *stopping_criterion* determines whether the optimal refinement leads to a good split based on the statistics associated with the optimal refinement and its complement. In our case, we compare *stopping_criterion* that user inputs with the highest information of refinements. If *stopping_criterion* is less than or equal to the highest information, then we will stop splitting on the considering branch.

3.5 Multi-Relational Decision Tree in Classification Process

In the first step (learning process), we build a multi-relational decision tree, which was explained in the previous section 3.4. In the second step (classification process), we are going to use the tree as classifier. In this classification process, the structure of the multi-relational decision tree after learning, is stored as a table in the database. The table that stores the built tree has the structure shown in Table 3.12.

The SQL query for the leaf nodes (represented by a value of -1 in the LEFT_CHILD and RIGHT_CHILD) is executed to set the value for the *class_label*. We set to majority class label in result of SQL query. The Algorithm 6 shows the pseudocode.

When we use the multi-relational decision tree as classifier, we have two types of classification: classify a new instance, and classify a new set of instances (or a new database that has the same

Columns	Meaning
NODE_ID	identification of node (0, for the root node).
SQL_QUERY	stores the SQL statement of the selection graph.
LEFT_CHILD	identification of the left child node.
RIGHT_CHILD	identification of the right child node.
CLASS_LABEL	stores the classification label, only for leaf nodes.

Table 3.12: Table structure storing the learned tree

Algorithm 6 SET_CLASS_LABEL (T)

```
1: Input: Table stored tree structure: T
2: for each row in T has LEFT_CHILD=-1 and RIGHT_CHILD=-1 do
3:    count total instances from SQL_QUERY;
4:    count positive instances from SQL_QUERY;
5:    if (positive ≥ total/2) then
6:       CLASS_LABEL := 'Y';
7:    else
8:       CLASS_LABEL := 'N';
9:    end if
10: end for
```

structure as the training database). These two classification tasks are described in the following sections.

3.5.1 Classify a New Instance

This classification uses depth-first search on multi-relational decision tree. We start searching from the root node of the tree. If the new instance is covered by the subset of instances returned from the SQL query corresponding to this node, we go through the left child node. Then, we check the existence of the new instance in the subset of the left child node. If the instance is covered, we will continue going to left branch, otherwise we go to the right node, and check if the instance is covered. We will stop when the node is a leaf node, and assign the *class_label* of this leaf node to the new instance. The Algorithm 7 shows the pseudocode for this type of classification.

Algorithm 7 CLASSIFY_INSTANCE ($\mathcal{T}.\mathcal{R}$, I)

```
1: Input: Root node of tree T.R, new Instance I.
2: Output: Class_label.
3: if T.R.L = true then
4:    return T.R.C;
5: else
6:    Sub:= Sem(T.R.LC.V);
7:    if I is in Sub then
8:       CLASSIFY_INSTANCE(T.R.LC, I);
9:    else
10:      CLASSIFY_INSTANCE(T.R.RC, I);
11:   end if
12: end if
```

3.5.2 Classify a New Database

The classification process described previously only covers the case of a new instance, but it can be the case that we have not only one but many new instances. Also, these new instances may have associations among other tables. In this case we will have classification of a new database. Given a new set of previously unseen instances to be classified, the queries of leaf nodes are applied to the database. The set of instances that a query returns will be assigned the class label that the corresponding selection graph has. According to the proposition 3, a given instance will not assign conflicting class labels. The Algorithm 8 is pseudocode for classification of new database.

Algorithm 8 CLASSIFY_DATABASE ($\mathcal{T}$, D)

1: *Input: tree $\mathcal{T}$, new Database D.*
2: *Output: Class_label for all instances in database.*
3: **for** each node n in $\mathcal{T}.\mathcal{N}$ **do**
4: **if** $n.L$=true **then**
5: CL:= $n.C$;
6: SI:=Sem($n.V$);
7: Given $D.T_0.ID \in SI$, set $D.T_0$.target_attribute = CL;
8: **end if**
9: **end for**

In multi-relational decision tree, each node represents one SQL query. In both of the above algorithms, the SQL query is the main item our algorithms spend time running it. Therefore, we try to reduce the number of SQL query running times. This is the reason that we use two different methods for two cases of classification:

- When classifying for an instance, we used Algorithm 7. Because most of binary trees have depth degree less than the number of leaf nodes. In Algorithm 7, the total time of SQL query running is the depth degree from root node to a particular leaf node.
- When classifying for a database, if we use Algorithm 7 then we have to apply the algorithm for each instance, sequentially. Therefore, the total time of SQL query running = number of instances * depth degree. Besides, if we use only leaf nodes to classify (corresponding to Algorithm 8), the total time of SQL query running = number of leaf nodes. Normally, the number of leaf nodes is almost less than the number of instances. Therefore, the new database is classified faster.

3.6 Multi-Relational Decision Tree in Practice

The system was designed according to the client/server architecture. It consists of three parts: the graphic user interface (GUI), the inference engine and the database connection. The system was implemented in Java* programming language.

*Java Technology website: http://java.sun.com/

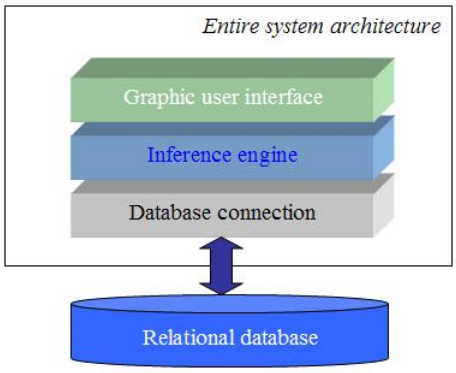


Figure 3.29: Overview of system architecture

3.6.1 Graphical User Interface

The graphical interface is used to interact with the mining engine to perform test and evaluation. It is built in order to have minimum intervention during the mining process. The evaluator or user initially uses the interface to connect to the desired database. The graphical interface consists of three functions: *Parameter*, *Learning* and *Classification*. The *Parameter* function helps the user to set the initial parameter values to start the system. The *Learning* function provides to the user the menu to interact with the inference engine to build the multi-relational decision tree. The *Classification* function supports the user to use the built tree as classifier. Figure 3.30 shows the main graphical user interface and the *Parameter* function.

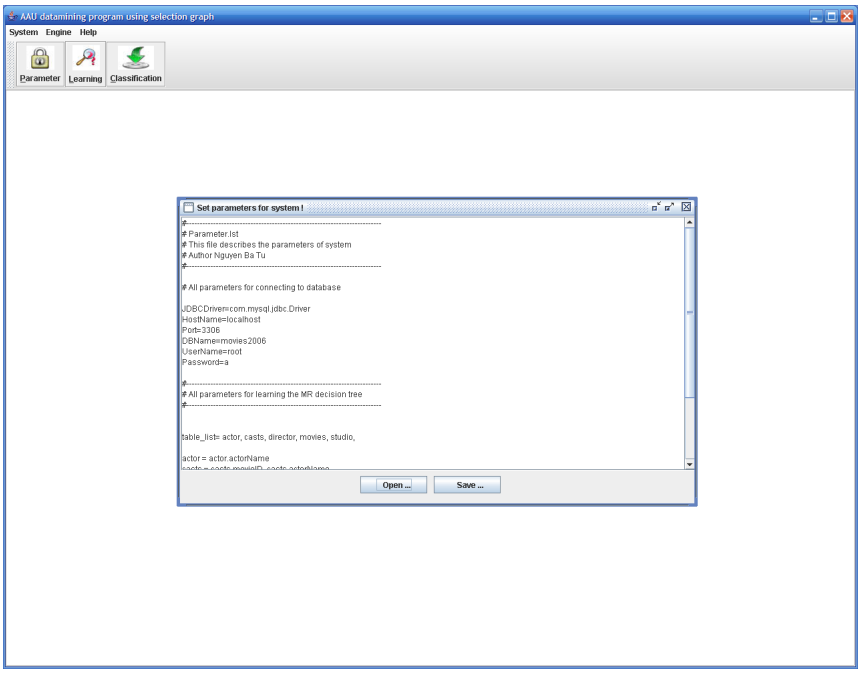


Figure 3.30: Main graphical user interface and 'Parameter' function

The Inference Engine

This part includes two phases:

1. Build the multi-relational decision tree. This phase is implemented according to the theory described in section 3.4.
2. Use the multi-relational decision tree as classifier.

3.6.2 Building the Multi-Relational Decision Tree

We implemented the multi-relational decision tree as binary tree. This tree consists of nodes and edges. The data structure of node includes five properties: NodeID, Value, Leaf, Right child node, and Left child node. Within a node, the Value property stores the identify of its selection graph. Leaf is used to check whether node is leaf node or not. Right child node and left child node are pointers to refer to the children node. The structure of multi-relational decision tree is shown in Figure 3.31.

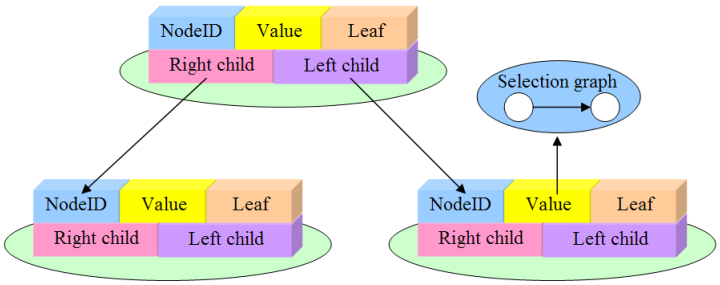


Figure 3.31: Data structure of multi-relational decision tree

In order to build the multi-relational decision tree, the GUI will first show some options for the user. Then the user has to choose the following parameters:

- Target attribute and target table.
- List of the interesting attributes.
- Input value for stopping criterion.

Based on Algorithm 5, we start with a selection graph including a single node at the root of the tree which represents the target table T_0. By analyzing all possible refinements of the considering selection graph, and examining their quality by applying information gain calculated in section 3.4.4, we determine the optimal refinement. This optimal refinement, together with its complement, are used to create the patterns associated with the left and the right branch respectively. Based on the stopping criterion it may turn out that the optimal refinement and its complement do not give cause for further splitting, a leaf node is introduced instead. Whenever the optimal refinement does provide a good split, a left and right branch are introduced and the procedure is applied to each of these recursively. The Figure 3.32 shows the Learning function from system.

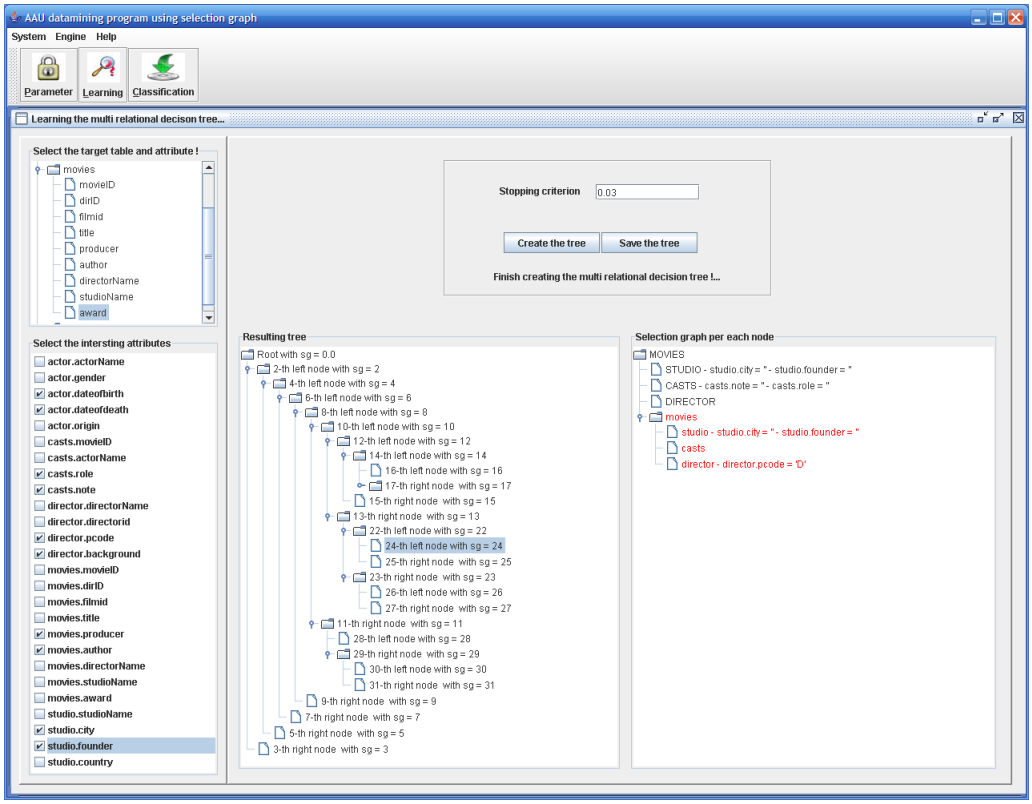


Figure 3.32: Interface of the ‘Learning’ function

In the Figure 3.32, the multi-relational decision tree consists of 31 nodes. Each node corresponds to one selection graph. The tree is shown in the left tree view, the corresponding selection graph is shown in the right tree view. The non-extended node of selection graph is presented in lowercases. The extended node of selection graph is presented in uppercases.

3.6.3 Using the Multi-Relational Decision Tree as Classifier

As described in section 3.5, this function will get SQL query and class_label from table in database. And then, It will run SQL query to get instances and apply class_label for each instances.

Figure 3.33 is an example from our system when classifying a new database.

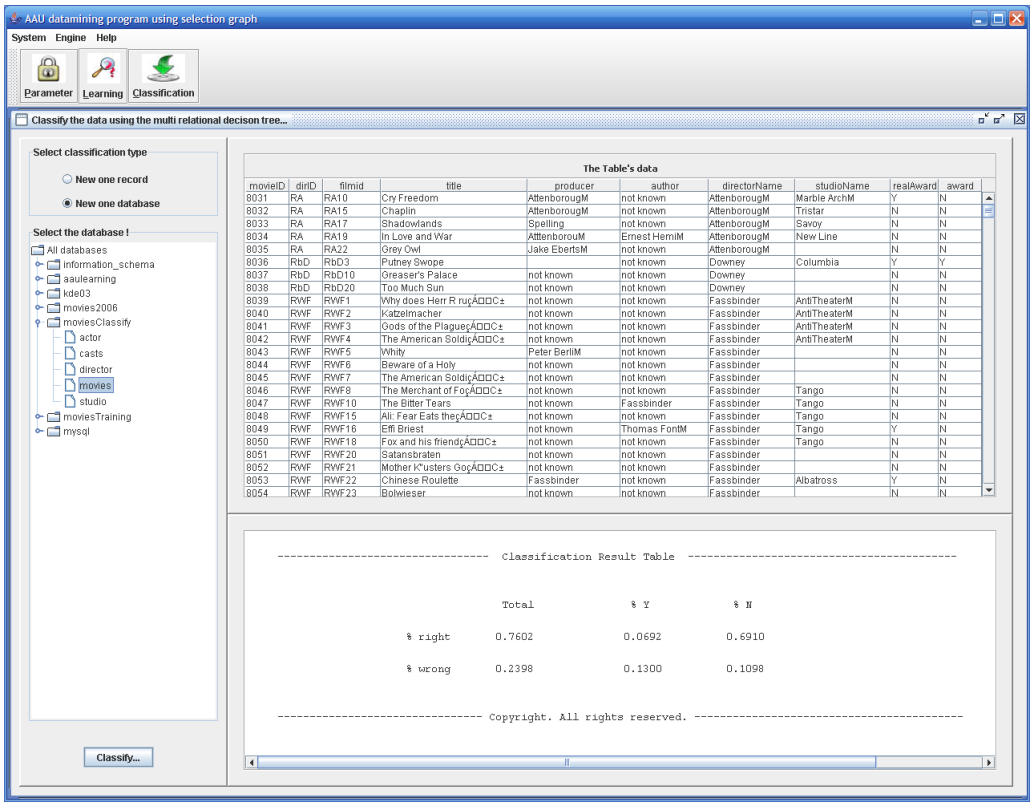


Figure 3.33: Interface of the ‘Classification’ function

Chapter 4

Experimental Results

In this chapter we show the results obtained after testing the system. First, we will refer to the database already introduced and used previously, the MOVIES database. Second, we will use another database for more tests, the FINANCIAL database. Finally, we will compare the results obtained from both databases with another software. The experiments were performed on a standard desktop PC (Pentium 4 CPU 2.66 GHz, 512 MB RAM) using MySQL* database running on the same machine.

4.1 MOVIES Database

As a demonstration of how our system works, we perform a number of experiments with the previously mentioned MOVIES database.

Task Description

The MOVIES database is a relational database with stored data about movies and related information. The tables in the database are ACTOR, STUDIO, MOVIES, CASTS, and DIRECTOR. The MOVIES table is the central table with each movie having a unique identifier. The actors of a given movie are stored in the CASTS table. Each row in the CASTS table represents the relationship between an actor and given movie in each cast. More information about individual actors is in ACTOR table. All directors in MOVIES table are listed in the table called DIRECTOR. Table STUDIO provides some information about the studios in which movies were made. The entity relationship diagram of this database is shown in Figure 4.1.

This database contains description of Movies and the characteristic to be predicted is represented by the attribute *Award*. The total number of tuples in the MOVIES table are 12046, the database also consists of 6714 actors, 186 studios and 3133 directors.

The main goal is to classify whether a movie received or not an award. The database has a column(award) that stores this attribute as Y if the movie received an award, N otherwise.

*MySQL website: http://dev.mysql.com/

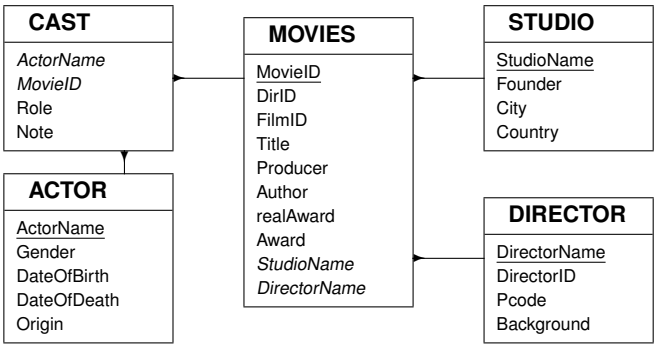


Figure 4.1: MOVIES database schema

Methodology

The database was divided into two disjoint subsets for the purpose of learning and classification process, respectively. The learning process used $\frac{2}{3}$ of instances. The other $\frac{1}{3}$ of the instances is used in the classification process. The process of dividing the database involved the splitting of not only the MOVIES table but all the related tables, since it is a relational database. The following Algorithm 9 shows the procedure for splitting the database.

Algorithm 9 SPLITTING_DATABASE (D, D_l, D_c)

1: *Input: Original Database D.*
2: *Output: Learning Database D_l, Classification Database D_c.*
3: Creating D_l, D_c so that they have the same structure with D;
4: /* `Creating the training database` D_l `*/`
5: Insert $\frac{2}{3}$ of D.MOVIES into D_l.MOVIES;
6: Insert D.DIRECTOR into D_l.DIRECTOR so that D.DIRECTOR.Directorname = D_l.MOVIES.Directorname;
7: Insert D.STUDIO into D_l.STUDIO so that D.STUDIO.StudioName = D_l.MOVIES.Studioname;
8: Insert D.CAST into D_l.CAST so that D.CAST.MovieID = D_l.MOVIES.MoviesID;
9: Insert D.ACTOR into D_l.ACTOR so that D.ACTOR.Actorname = D_l.CAST.Actorname;
10: /* `Creating the classification database` D_c `*/`
11: D_c.MOVIES := D.MOVIES minus D_l.MOVIES;
12: Do line 6 to 9 again with D_c instead of D_l;
13: **return** D_l, D_c

During the learning process, we used different stopping criteria and selected different interesting attributes. Although, the system was run with different selection of interesting attributes, different input for stopping criterion, but always the same target table and target attribute: MOVIES and award, respectively. The results varied in size of the obtained tree (number of nodes), and in the accuracy obtained in the classification process. Further explanation on how the tests were performed and the results obtained are presented in the following sections.

Results

The ***first test*** we performed consisted in selecting all the attributes from our list of interesting attributes. We keep that list but change the input value of the stopping criterion. The purpose of this test is to find the *optimal* value (or interval) for the stopping criterion. We consider we have an optimal value when we get the highest accuracy.

stopping criterion	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
9×10^{-15}	57	76.39	124.29	3.93
⋮	⋮	⋮	⋮	⋮
9×10^{-14}	57	76.39	128.73	3.73
⋮	⋮	⋮	⋮	⋮
9×10^{-13}	57	76.39	127.46	3.01
⋮	⋮	⋮	⋮	⋮
9×10^{-12}	57	76.39	128.98	2.62
⋮	⋮	⋮	⋮	⋮
9×10^{-11}	57	76.39	126.31	2.79
⋮	⋮	⋮	⋮	⋮
9×10^{-10}	57	76.39	128.92	2.68
⋮	⋮	⋮	⋮	⋮
9×10^{-9}	57	76.39	127.07	2.74
⋮	⋮	⋮	⋮	⋮
9×10^{-8}	57	76.39	128.82	2.79
⋮	⋮	⋮	⋮	⋮
9×10^{-7}	57	76.39	124.98	3.02
⋮	⋮	⋮	⋮	⋮
9×10^{-6}	57	76.39	126.15	3.17
⋮	⋮	⋮	⋮	⋮
9×10^{-5}	57	76.39	127.85	3.10
⋮	⋮	⋮	⋮	⋮
9×10^{-4}	57	75.92	121.61	3.04
⋮	⋮	⋮	⋮	⋮
9×10^{-3}	49	75.65	74.09	2.92
⋮	⋮	⋮	⋮	⋮
9×10^{-2}	3	69.00	4.25	1.79

[a] Learning process
[b] Classification process

Table 4.1: Results with different stopping criteria and the same attributes for MOVIES

We are interested in measuring the number of nodes, the time it takes to create the tree, the time in the classification process and the accuracy when classifying. Table 4.1 shows these results, obtained changing the input for the stopping criterion but using always the same list of interesting

attributes. The interesting attributes used for this test were all attributes in the database except the target attribute (the 25 attributes from Table 4.2).

attributes *(Table.column)*	**parameters** *(#)*
Movies.year Movies.producer Movies.author Movies.directorName Movies.studioName	5
Studio.studioName Studio.city Studio.country Director.directorName Director.background	10
Casts.movieID Casts.actorName Casts.role Actor.actorName Actor.gender	15
Studio.founder Director.pcode Casts.note Actor.dateOfBirth Actor.dateOfDeath	20
Movies.dirID Movies.filmID Movies.title Actor.origin Actor.notes	25

Table 4.2: List of selected attributes for MOVIES

Based on the results from the previous test we can set an input for the stopping criterion for our ***second test***, but now we will change the list for the interesting attributes. We will start with a few and increase the number until we have all of them. The purpose of this test is to find the best result and evaluate the relationship between the accuracy and the attributes. Firstly, we choose the value of stopping criterion to be 1×10^{-6}, then 5×10^{-6} and finally 9×10^{-6}. With each stopping criterion, we change the list for the interesting attributes as shown in Table 4.2.

In Table 4.3 we can see the results of the test in terms of size of the obtained tree, accuracy achieved with the classifier and time. We used stopping criterion to be 1×10^{-6}. For this tests we changed the number of attributes, starting with a few and increasing the number, but we always used the same stopping criterion.

Table 4.4 shows the results after applying the same selection of attributes as above but now with a stopping criterion of 5×10^{-6}. In Table 4.5 we show the results with a stopping criterion of 9×10^{-6}.

We choose the values of stopping criterion in the optimal interval but change the list of attributes. We choose the values of stopping criterion to be (9×10^{-6}, 5×10^{-6}, 1×10^{-6}), and we get the similar accuracy for all the tests. The reason is that the stopping criteria we choose, is the threshold. Hence, the accuracy does not change and is always equal to the best value. In order to

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
5	55	76.39	113.96	3.93
10	59	76.39	130.06	3.04
15	59	76.39	130.06	4.21
20	57	76.39	134.67	4.95
25	57	76.39	124.04	4.68

[a]Learning process
[b]Classification process

Table 4.3: Results with increasing number of attributes and stopping criterion=1×10^{-6}

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
5	55	76.39	107.43	4.28
10	59	76.39	125.39	4.26
15	59	76.39	123.34	4.50
20	57	76.39	127.10	2.71
25	57	76.39	130.65	3.76

[a]Learning process
[b]Classification process

Table 4.4: Results with increasing number of attributes and stopping criterion=5×10^{-6}

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
5	55	76.39	105.75	3.37
10	59	76.39	124.21	4.06
15	59	76.39	126.76	4.06
20	57	76.39	122.32	2.31
25	57	76.39	123.35	2.26

[a]Learning process
[b]Classification process

Table 4.5: Results with increasing number of attributes and stopping criterion=9×10^{-6}

estimate the important degree of attributes to the accuracy, we choose the stopping criteria that do not belong to the optimal interval. These results are shown below.

In Table 4.6 we can see the results of the test in terms of size of the obtained tree, accuracy achieved with the classifier and time. For this tests we changed the number of attributes, starting with a few and increasing the number, but we always used the same stopping criterion of 0.001.

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
5	51	76.34	89.21	2.32
10	57	76.39	117.15	2.92
15	57	76.39	115.56	2.76
20	57	76.39	128.36	2.78
25	57	75.92	118.85	2.98

[a]Learning process
[b]Classification process

Table 4.6: Results with increasing number of attributes and stopping criterion=0.001

Table 4.7 shows the results after applying the same selection of attributes as above but now with a stopping criterion of 0.005. In Table 4.8 we show the results with a stopping criterion of 0.009.

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
5	55	76.39	115.51	2.15
10	55	76.00	146.59	2.23
15	55	75.57	91.39	2.64
20	53	75.10	89.87	2.61
25	55	75.10	100.26	3.02

[a]Learning process
[b]Classification process

Table 4.7: Results with increasing number of attributes and stopping criterion=0.005

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
5	41	76.22	96.20	2.42
10	53	76.00	151.68	2.92
15	35	75.57	56.81	2.23
20	35	75.57	57.21	1.76
25	41	75.57	70.35	1.82

[a]Learning process
[b]Classification process

Table 4.8: Results with increasing number of attributes and stopping criterion=0.009

Table 4.9 shows the best results for the classification process. These results are obtained by comparing the class label that the classifier assigns to each of the instances with a real class value stored in the database.

		True	
		Y	**N**
Predicted	Y	10.93	14.72
	N	8.89	65.46

Table 4.9: Confusion matrix for MOVIES database

Models

We have previously presented the general results when testing the system. In this part we present some models and selection graphs to get and idea on how the trees look like.

Figure 4.2 shows part of one model of the trees obtained during the learning process. In this top part of the tree we can notice that the nodes in the tree have selection graphs with only conditions on the target table (MOVIES). The interesting part from this tree is that we can see that it splits on the condition 'studioname='', meaning that a movie must have something as studio, and from there other conditions are added to the refinements of selection graphs in the tree. It is important to mention that as the tree grows, not only conditions are added but also edges.

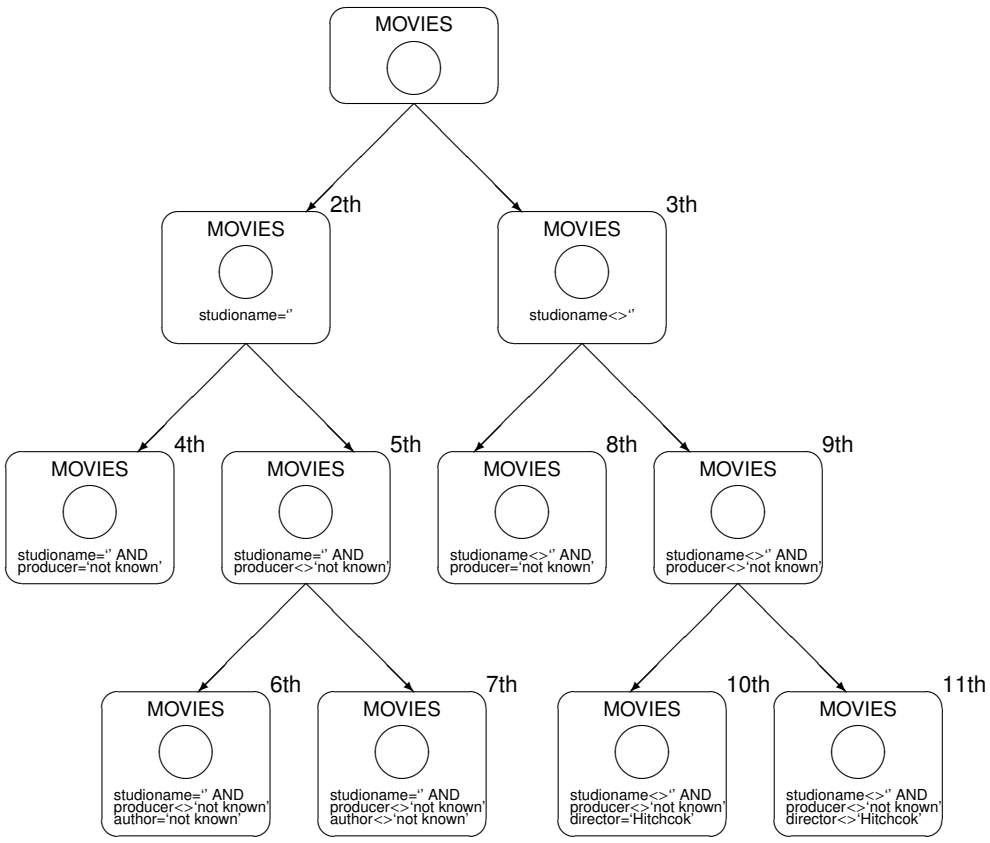


Figure 4.2: Resulting tree for MOVIES

Figure 4.3 shows one example of a selection graph and its complement obtained from a leaf node in the resulting tree. We can see that this selection graph consists of two nodes with conditions on both nodes.

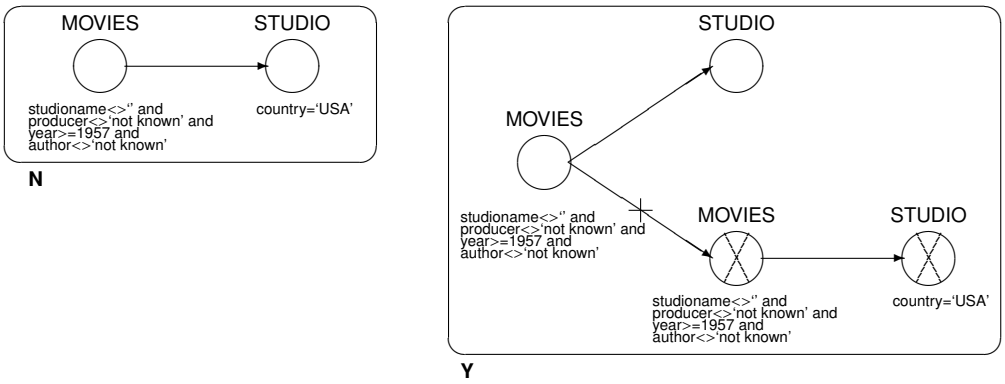


Figure 4.3: One example of a selection graph obtained in the tree for MOVIES

In Figure 4.4 we show another example of a more complex selection graph and its compliment. This selection graph corresponds to another leaf node in the resulting tree.

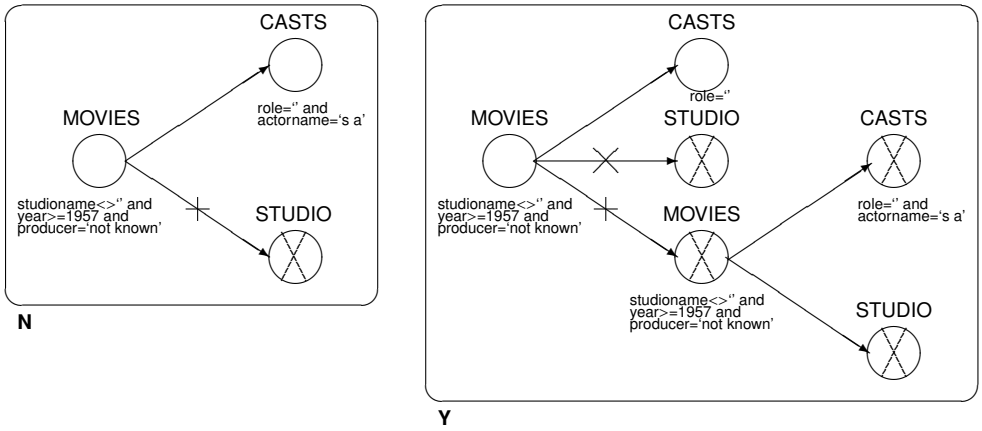


Figure 4.4: Another example of a selection graph obtained in the tree for MOVIES

In both cases (Figure 4.3 and Figure 4.3), we can see that not only the target table (MOVIES) is present in the nodes and that conditions are also added for the other nodes in the selection graphs.

Analysis

- The best accuracy observed was around 76%
- We started the testing with all attributes and different stopping criteria. We got the optimal interval of stopping criteria between 9×10^{-5} and 9×10^{-15}.
- In order to estimate the important degree of attributes to the accuracy, we choose the stopping criteria that do not belong to the optimal interval. And then, we had the result that was shown in Table 4.6 with 0.001, Table 4.7 with 0.005, and Table 4.8 with 0.009. Based on these results, we observe that the attributes have a different important degree in the accuracy. For example, in Table 4.8 with 0.009, when the number of attributes is equal to 5 or 10, we obtain the better accuracy than the one with 15, 20 and 25 attributes. We think that some attributes are importance and others can make the noise. This proves that attributes have a different important degree in relation to the accuracy.

4.2 FINANCIAL Database

At present, banks offer different services to individuals. These services include managing account, credits, loans and others. The bank stores data about their clients, the accounts (transactions within several months), the loans already granted, the credit cards issued in a relational database. The following Figure 4.5 shows the financial database used in PKDD'99 Discovery Challenge [12].

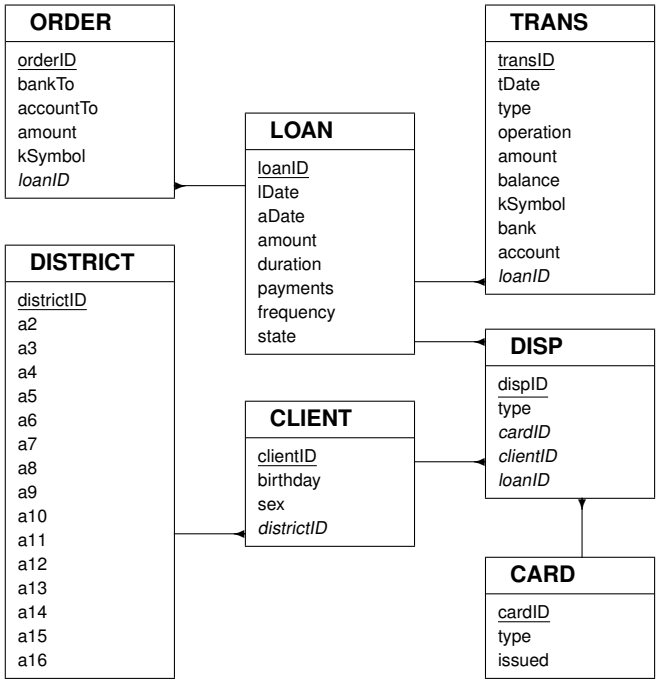


Figure 4.5: FINANCIAL database schema

The database consists of seven related tables: TRANS, DISTRICT, LOAN, CLIENT, ORDER, DISP and CARD. The meaning of each table is shown as follows:

CLIENT	characteristics of a client.
DISP	relates together a client with a loan.
	(i.e. this relation describes the rights of clients to apply loans).
ORDER	characteristics of a payment order.
TRANS	describes one transaction on a loan.
LOAN	describes a loan granted for a given client.
CARD	a credit card issued to a client.
DISTRICT	describes demographic characteristics of a district.

In this database, the table CLIENT describes characteristics of persons who can manipulate with the loans. The table LOAN and CARD describe some services which the bank offers to its clients; more credit cards can be issued to a client, at most one loan can be granted to a client. Clients, credit cards and loans are related together in the relation 'DISP' (disposition). The table DISTRICT gives some publicly available information about the districts (e.g. the unemployment rate); additional information about the clients can be deduced from this table.

This database consists of 682 loans, 827 clients, 54694 transactions, 1513 orders, 827 dispositions, 36 cards and 77 districts.

Task Description

The goal is to classify whether a loan is granted (Y), or not (N). The database has a column (state) that stores this attribute.

Methodology

The database was divided into two disjoint subsets for the purpose of learning and classification process, respectively. The learning process used $\frac{2}{3}$ of instances. The other $\frac{1}{3}$ of the instances is used in the classification process. The process of dividing the database involved the splitting of not only the loan table but all the related tables, since it is a relational database. A procedure similar to Algorithm 9 was applied to this database.

Results

The ***first test*** we performed consisted in selecting 15 attributes from our list of interesting attributes shown in Table 4.10. We keep that list but change the input value of the stopping criterion. The purpose of this testing is to find the *optimal* value (or interval) for the stopping criterion. We consider we have an optimal value when we obtain the highest accuracy.

attributes *(Table.column)*	**parameters** *(#)*
Loan.duration Loan.payments Loan.frequency	3
Client.sex Card.type Disposition.type	6
Order.bank_to Order.k_symbol Transaction.type	9
Transaction.operation Transaction.k_symbol Transaction.bank	12
District.a2 District.a3 District.a4	15

Table 4.10: List of selected attributes for FINANCIAL

Table 4.11 shows the results after running the program with a fixed 15 attributes but changing the stopping criteria.

Based on the results from the previous test we can set an input for the stopping criterion for our ***second test***, but now we will change the list for the interesting attributes. We will start with a few and increase the number until we have all of them. The purpose of this test is to find the best result and evaluate the relationship between the accuracy and the interesting attributes.

stopping criterion	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
1×10^{-5}	27	76.21	75.47	1.79
⋮	⋮	⋮	⋮	⋮
1×10^{-4}	27	76.21	76.16	1.88
⋮	⋮	⋮	⋮	⋮
1×10^{-3}	27	76.21	75.27	1.70
⋮	⋮	⋮	⋮	⋮
1×10^{-2}	23	76.21	64.75	1.67
⋮	⋮	⋮	⋮	⋮
1×10^{-1}	3	76.21	8.45	1.73

[a]Learning process
[b]Classification process

Table 4.11: Results with different stopping criteria with same attributes for FINANCIAL

Table 4.12 shows the results obtained after testing the system with the interesting attributes according to Table 4.10 and using the same stopping criterion, in this case 1×10^{-1}.

attributes *(#)*	**size** *(# nodes)*	**accuracy** *(%)*	**time**[a] *(sec)*	**time**[b] *(sec)*
3	25	76.21	66.83	1.75
6	27	76.21	76.24	1.78
9	27	76.21	75.42	2.17
12	27	76.21	75.56	2.34
15	27	76.21	75.47	2.61

[a]Learning process
[b]Classification process

Table 4.12: Results with increasing number of attributes and stopping criterion=1×10^{-1}

Table 4.13 shows the best results for the classification process. These results are obtained by comparing the class label that the classifier assigns to each of the instances with a real class value stored in the database.

		True	
		Y	**N**
Predicted	Y	26.43	12.33
	N	11.45	49.78

Table 4.13: Confusion matrix for FINANCIAL database

Models

We have previously presented the general results when testing the system. In this part we present some models and selection graphs to get and idea on how the trees look like. The Figure 4.6 shows one tree obtained after we run our tests.

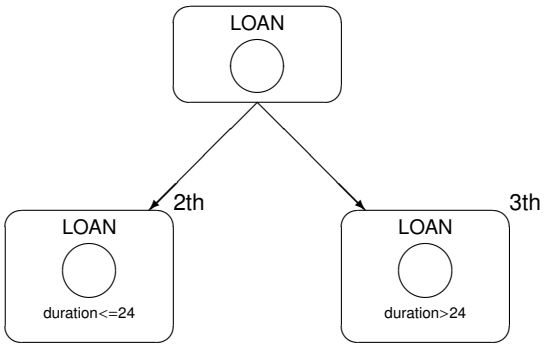


Figure 4.6: Resulting tree for FINANCIAL

Figure 4.7 shows one selection graph and its complement corresponding to one leaf node in the resulting tree. The selection graph consists of two nodes with conditions on both nodes. Also, we can see as the tree grows both edges and conditions are added to the nodes.

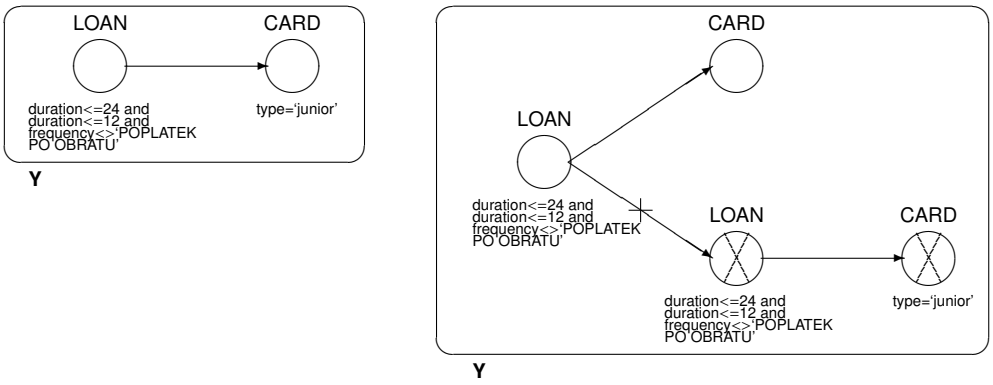


Figure 4.7: Example of selection graph with its complement obtained in the tree

Analysis

- The only accuracy observed was 76.21%
- We started the testing with 15 attributes and different stopping criteria. We got the optimal interval of stopping criterion is 1×10^{-1} to 1×10^{-5}.
- Based on Table 4.12, we used three attributes in table LOAN as input, we observed that the duration attribute is the most important. Because we can get the best accuracy and this attribute is used as unique variable in model of this multi-relational decision tree.

4.3 Comparison with Clementine

Clementine® [†] is a data mining tool for the quickly development of predictive models using business expertise to improve decision making. It is designed around the industry-standard CRISP-DM model, and supports the entire data mining process, from data to better business results.

[†]Clementine website: http://www.spss.com/clementine/index.htm (we used version 9.0)

In this section we compare our previous results with Clementine. The main idea is to show that our system performs as good as other system, having only one table for consideration.

4.3.1 MOVIES

In this part we compare the results obtained from the MOVIES database with Clementine. The test script is built as follows:

- We use only table MOVIES (title, producer, author, directorname, studioname, and award) for both testing on Clementine and our implementation.
- We use modeling for decision tree based on the algorithm C5.0 in Clementine.
- We run our implementation with different stopping criteria.
- We choose the best result from our implementation.
- We compare the tree structure and accuracy between the two systems.

Below, we describe our testing in detail. First, we show the testing on Clementine and then on our implementation.

Clementine Model

In Figure 4.8, we show the model we built. Clementine supports several modelings to build decision trees: C5.0, CHAID, and C&R, but we only use the modeling C5.0 to build the decision tree. In this model, we used the MOVIES table as the data source, therefore attributes from that table are our input parameters including *title, producer, author, directorname, studioname*. We also define *award* as our output parameter. In Clementine, they support three methods (First, 1-to-n, Random %) to split dataset. We choose the First method to split data because it can create the training and testing dataset that are similar to datasets we used in our implementation. We used a sample node for dividing the table in two different sets training and testing. For the training set we selected the first $\frac{2}{3}$ of the total records in the table. For the testing set we discarded the same $\frac{2}{3}$ of the sample used for training.

Finally we build the tree on the training set and get a model for each case. The resulting tree structure is shown in Figure 4.9.

After running our model in the training set, we can analyse it obtaining the summary and accuracy showed in Figure 4.10.

MRDT Implementation

In this testing, we only use attributes in table MOVIES including *title, producer, author, directorname, studioname* as input, and *award* as output. We also use stopping criterion to be 9×10^{-5}. We choose this number because it belongs to the optimal interval of stopping criterion. During the learning process, we get the decision tree with 11 nodes, shown in Figure 4.11.

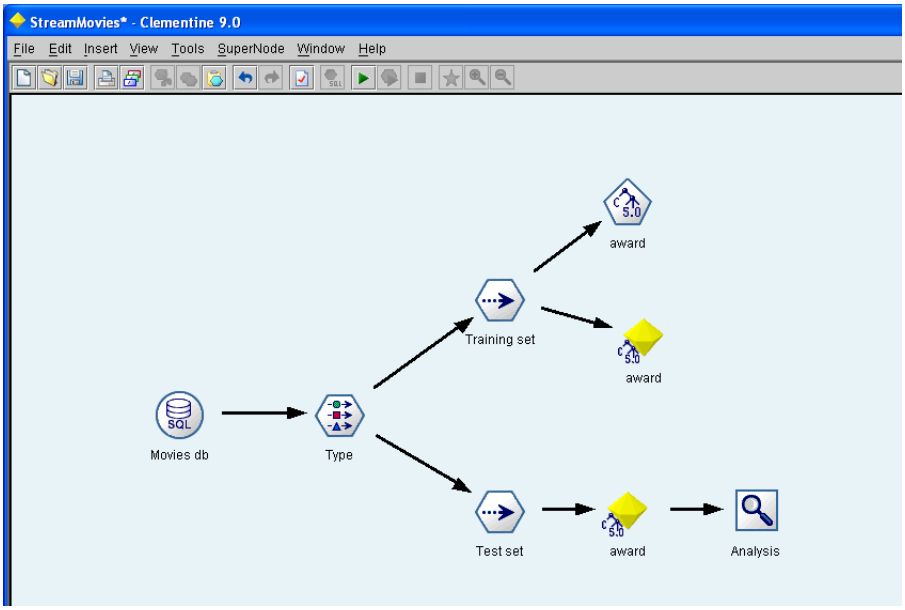


Figure 4.8: Modeling in Clementine for MOVIES

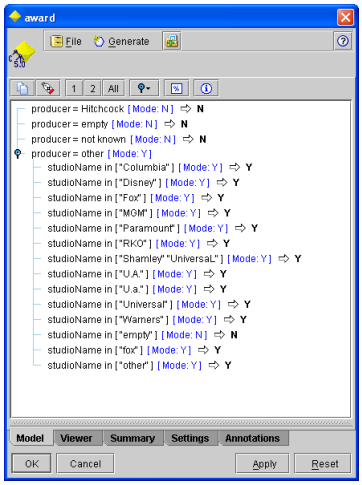


Figure 4.9: Decision tree for MOVIES drawn from Clementine

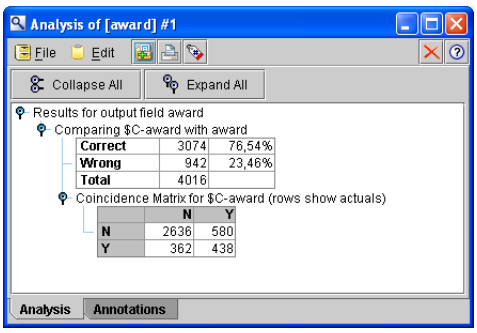


Figure 4.10: Analysis obtained with Clementine for MOVIES

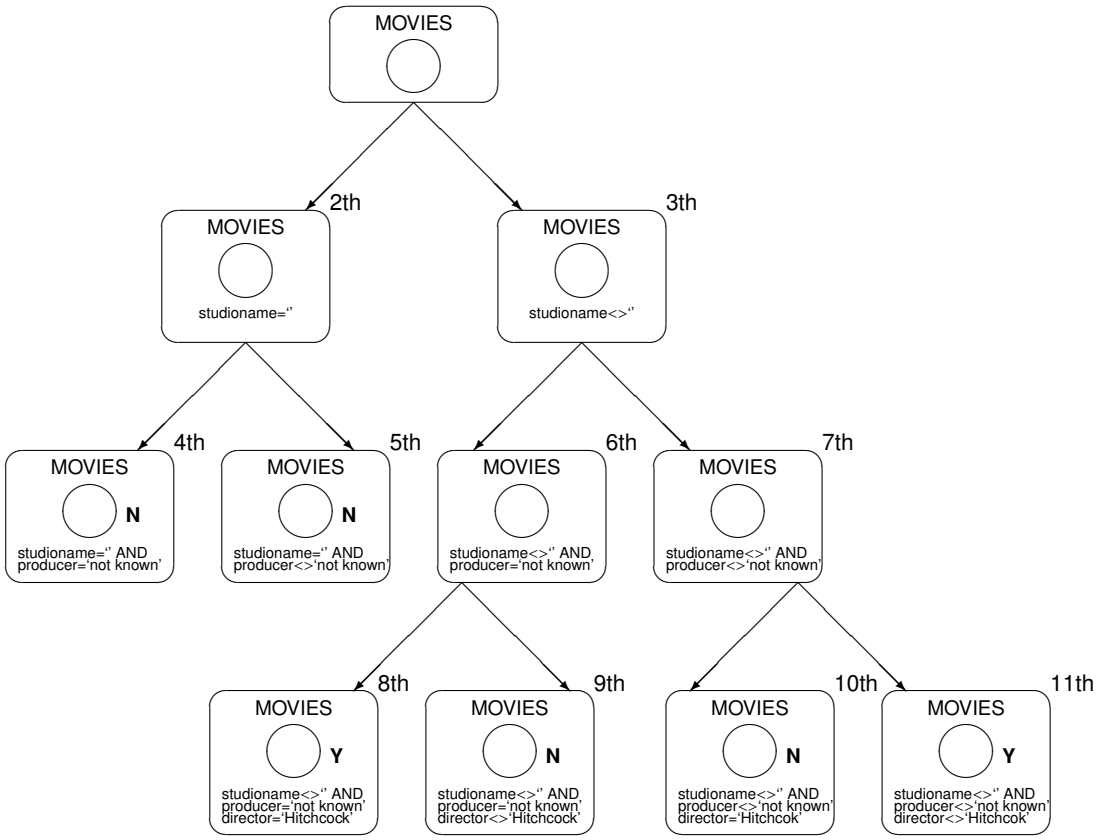


Figure 4.11: Resulting tree obtained using MOVIES table

And the obtained accuracy is 76,39% in the classification process. The confusion matrix we obtained is shown in Table 4.9.

Comparison

Based on the experimental results performed in previous section, we can conclude that:

- Both accuracies are the similar (one is 76,54% and other is 76,39%).
- Both tree structures are different. In Clementine, the depth of decision tree is 2 and consists of 18 nodes. In our implementation, the depth of decision tree is 4, and includes 11 nodes. We have a different structure of decision tree, because we use different learning algorithms. Therefore, the order of attributes in the two models is different. In Clementine, producer attribute is chosen before studioname, while our implementation reverses and continues with director attribute.
- We obtained different trees but similar accuracies, because both trees choose the attributes: producer, studioname as important variables for classification.
- We obtained a good result from our implementation.

4.3.2 FINANCIAL

In this part we compare the results obtained from the FINANCIAL database with Clementine. The test script is built as follows:

- We use only table LOAN (amount, duration, payments, and state) for both testing on Clementine and our implementation.
- We use modeling for decision tree based on the algorithm C5.0 in Clementine.
- We run our implementation with different stopping criteria.
- We choose the best result from our implementation.
- We compare the tree structure and an accuracy between the two systems.

Below, we describes our testing in detail. First, we shows the testing on Clementine and then on our implementation.

Clementine Model

Once again, in Clementine we only use the modeling C5.0 to build the decision tree. In Figure 4.12, we show the model we built.

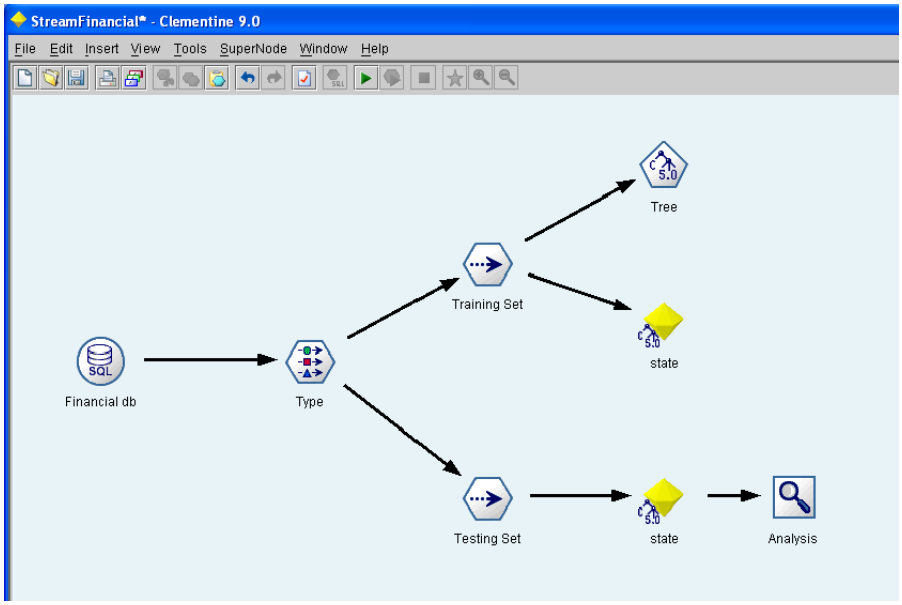


Figure 4.12: Modeling in Clementine for FINANCIAL

In this model, we used the LOAN table as the data source, therefore attributes from that table are our input parameters including *amount, duration*, and *payments*. We also define *state* as our output parameter. We divided the records in two different sets (training and testing). We also used First method to split dataset. We used a sample node for dividing the table in two different sets training and testing. For the training set we selected the first $\frac{2}{3}$ of the total records, discarding the same $\frac{2}{3}$

for the testing set. Finally we build the tree on the training set and get a model for each case. The resulting tree structure is shown in Figure 4.13.

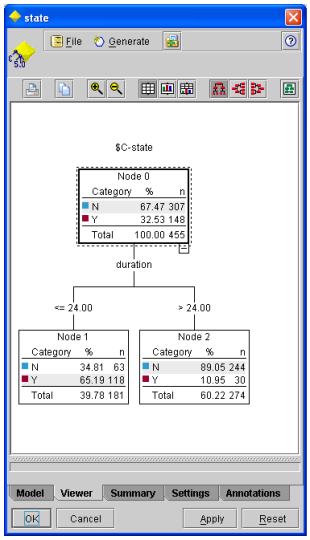


Figure 4.13: Decision tree drawn from Clementine for FINANCIAL

After running our model in the training set, we can analyse it obtaining the summary and accuracy showed in Figure 4.14.

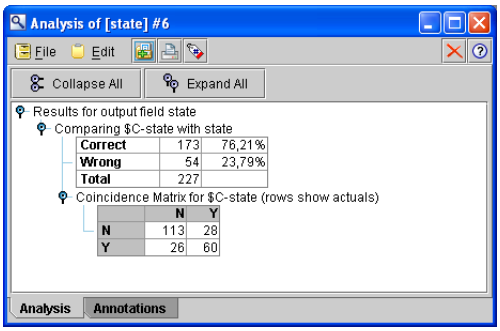


Figure 4.14: Analysis obtained with Clementine for FINANCIAL

MRDT Implementation

Based on testing results in section 4.2, we choose the optimal stopping criterion be 1×10^{-1}, and list of attributes in LOAN table including *amount, duration*, and *payments*. We have the tree structure in Figure 4.6. The confusion matrix we obtained is shown in Table 4.13.

Comparison

Based on the experimental results performed in previous section, we can conclude that: both tree structures and both accuracies are the same and we obtained a good result from our implementation.

Chapter 5

Conclusion

Multi-relational data mining has been studied for many years. This framework focuses on the discovery of useful information from relational database using the pattern language called selection graph. This framework has several advantages, but there are still a few disadvantages. To solve some of the disadvantages from previous definitions of selection graph, we introduced the formal definition of selection graph. Based on this definition, we constructed the multi-relational decision tree. And then, we implemented a multi-relational decision tree.

We have tested this multi-relational decision tree on two well-known relational databases, MOVIES and FINANCIAL. The experimental results were promising and showed that it is feasible to mine from relational databases. Also, we compared the results against a commercial tool for data mining, Clementine. The accuracy obtained in both cases was similar. The positive results suggest that the current work could be continued in future researches.

5.1 Further Work

Even though we will not conduct more research, we give some directions that could be taken from this point.

Aggregate Functions Implement the use of aggregate functions as an extension to the system. The purpose of using the aggregate functions is to deal with one-to-many relationships in the database. To use the aggregate functions, we have to extend the definition of selection graph. The edge in a selection graph, which originally represented existential associations, are annotated with aggregate functions that summarize the selected substructure by the connected subgraph. This means that whenever a new table is involved over a one-to-many association, an aggregate functions can be applied to capture its features. On the other hand, we can use aggregate functions to characterize the structural information that is stored in tables and associations between them. The detailed desciption on how this extension could be used can be found in [13].

Relational Features We based our work in selection graphs, but there is another approach known as relational features [14] when selecting the patterns. An extension to the system can be implemented so both approaches can be specified and compared. In propositional data, a

feature is defined as a combination of an attribute, an operator and a value. For example, a feature of Movies might be *producer = 'Stromberg'* - movies that its producer is 'Stromberg'. Relational feature is also a combination of an attribute, an operator and a value but the attribute is referenced by a feature that belongs to other related object. For example, a relational feature of Movies can be *Movie(x), Studio(y): country(y)='USA' and studioIn(x,y)* - determines whether the country of a studio that makes a movie is 'USA'. When object *X* has a one-to-many relation to object *Y*, relational feature must consider set of attribute values on the object *Y*. In this case, besides using the standard database aggregate functions to map set of values to single values, Neville and his colleague introduced DEGREE feature as the degree of objects and the degree of links. The detailed desciption on how this extension could be used can be found in [14].

Bibliography

[1] Heikki Mannila David J. Hand and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[2] Nada Dzeroski, Saso; Lavrac. *Relational Data Mining*. Springer, 2001.

[3] E. Simoudis. Reality check for data mining. In *IEEE Expert: Intelligent Systems and Their Applications*, volume 11, pages 26–33, 1996.

[4] Heikki Mannila. Methods and problems in data mining. In *ICDT*, pages 41–55, 1997.

[5] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.

[6] Arno J. Knobbe, Hendrik Blockeel, Arno P. J. M. Siebes, and D. M. G. van der Wallen. Multi-relational data mining. Technical Report INS-R9908, 31, 1999.

[7] Arno J. Knobbe, Arno Siebes, and Daniel van der Wallen. Multi-relational decision tree induction. In *Principles of Data Mining and Knowledge Discovery*, pages 378–383, 1999.

[8] Héctor Ariel Leiva. MRDTL: A multi-relational decision tree learning algorithm, 2002.

[9] Anna Atramentov and Vasant Honavar. Speeding up multi-relational data mining, 2003.

[10] A. Atramentov, H. Leiva, and V. Honavar. A multi-relational decision tree learning algorithm: Implementation and experiments, 2003.

[11] Henry F. Korth Abraham Silberschatz and S. Sudarshan. *Database System Concepts, 4th edition*. McGraw-Hill, 2002.

[12] PKDD '99 Discovery Challenge: A collaborative effort in knowledge discovery from databases. http://lisp.vse.cz/pkdd99/challenge/chall.htm, Seen May 16, 2006.

[13] A.J. Knobbe, A. Siebes, and B. Marseille. Involving aggregate functions in multi-relational search. August 2002.

[14] Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay. Learning relational probability trees. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 625–630, New York, NY, USA, 2003. ACM Press.