

Automatic Generation Of Hierarchical Structures from Large database

*Dissertation submitted to Jawaharlal Nehru University
In partial fulfillment of the requirement for the award
of degree of*

**MASTER OF TECHNOLOGY
In
COMPUTER SCIENCE & TECHNOLOGY**

By

Sudesh Singh Chandel

**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067**

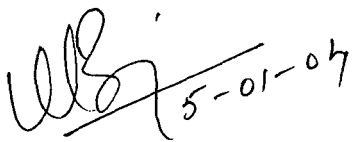
JANUARY 2004

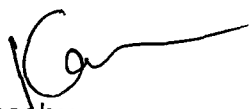
CERTIFICATE

This is to certify that the thesis entitled "**Automatic Generation of Hierarchical Structures from Large Databases**", being submitted by Mr. Sudesh Singh Chandel to **The School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi** in partial fulfilment of the requirement for the award of the degree of **Master of Technology in Computer Science & Technology**, is a record of original work done by him under the supervision of Prof. K K Bharadwaj during the Monsoon Semester, 2003.

The results reported in this thesis have not been submitted in part or full to any other University or Institution for the award of any degree etc.


Sudesh Singh Chandel
(Student)


Prof. KK Bharadwaj
(Supervisor)


Prof. Karmeshu
(Dean, SC&SS, JNU, New Delhi-110067)

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my supervisor Prof. K. K. Bharadwaj, for his enthusiasm, support and encouragement throughout my thesis. Without his consistent guidance and timely advice, the thesis would not have been completed.

I would like to thank all the staff of SC &SS, JNU especially to the caretaker of lab, where I spent most of my time working for the thesis. My thanks also goes to the staff members of the university library for providing the conducive environment to study for the research.

And finally special thanks to my family and friends who have been a constant source of support throughout my thesis.


Sudesh Singh Chandel

ABSTRACT

Data mining is the nontrivial extraction of implicit, previously unknown and potentially useful information from data. As one of the most important background knowledge, hierarchy plays a fundamentally important role in data mining. It is the purpose of this thesis to study some aspects of hierarchy and the automatic generation of hierarchical structures.

Concept hierarchies organize data and concepts in hierarchical forms or in certain partial order, which helps expressing knowledge and data relationships in databases in concise, high level terms and thus plays an important role in knowledge discovery processes. Concept hierarchies could be provided by knowledge engineers, domain expert, users or embedded in some data relations. However, it is sometimes desirable to automatically generate some concept hierarchies or adjust some given hierarchies for particular learning tasks.

As the number of rules becomes significant, they are not comprehensible to people as meaningful knowledge from which they can gain insight into the basis of decision making. The Hierarchical Production Rule (HPR) is a technique for restructuring production rules to generate comprehensible knowledge structure. An algorithm is developed that organizes the standard production rules into hierarchical structure that can further be used to generate Hierarchical Production Rules and also an algorithm for automatic generation of hierarchical structure from database is proposed. In the present work proposed algorithms is included and results are presented.

CONTENTS

	Page No.
1. Introduction	
1.1 DM: On what kind of data	1
1.2 Data Mining Functionality- What kind of patterns can be mined	6
2. Hierarchical Censored Production Rules / Hierarchical Production Rules	
2.1 Introduction	10
2.2 Hierarchical Censored Production Rules	12
2.3 HCPR-tree	14
2.4 Inference rules	17
3. Automatic Generation of Hierarchical Structure	
3.1 Introduction	20
3.2 Concept hierarchy and its role in knowledge discovery in databases	21
3.3 Concept hierarchy generation	
3.3.1 Generation of concept hierarchy for numerical data	23
3.3.2 Concept hierarchy generation for categorical data	28
3.4 Algorithm (Generation Of HPRs)	32
3.5 Algorithm (Automatic generation of hierarchical structure)	44
4. Implementation and Results	
4.1 Introduction	48
4.2 Generation of HPRs	48

	Page No.
4.3 Automatic generation of hierarchical structure	50
5. Conclusion	54
Bibliography	55

CHAPTER 1

INTRODUCTION

The work presented in this dissertation includes, an algorithm that organizes the standard production rules into hierarchical structure that can further be used to generate Hierarchical Production Rules and also an algorithm for automatic generation of hierarchical structures from databases.

In the present work proposed algorithms is included and results are presented. Chapter 1 introduces to the data-mining, On what data data-mining is applicable and what are the types of patterns that can be generated. The concept of Hierarchical Censored Production Rules (HCPRs) are explained in Chapter 2. Chapter 3 discusses the concept of automatic generation of hierarchical structures from databases, Both proposed algorithms are presented and explained in the chapter. Implementation of the algorithm and results obtained are presented in the chapter 4. Thesis ends with conclusion in Chapter 5.

1.1 DM: On what kind of data

Data mining is the nontrivial extraction of implicit, previously unknown and potentially useful information from data. [2]

The fast-growing, tremendous amount of data, collected and stored in large and numerous databases, has far exceeded our human ability for comprehension without powerful tools. As a result, data collected in large databases become "data-tombs" data archives that are seldom visited. Consequently, important decisions are often made based not on the information-rich data stored in databases but rather on a decision maker's intuition, simply because the decision maker does not have the tools to extract the valuable knowledge embedded in the vast amounts of data. In addition, consider current expert system technologies, which typically rely on users or domain experts to manually input knowledge

into knowledge bases. Unfortunately, this procedure is prone to biases and errors, and is extremely time consuming and costly. Data mining tools perform data analysis and may uncover important data patterns, contributing greatly to business strategies, knowledge bases, and scientific and medical research. The widening gap between data and information calls for a systematic development of data mining tools that will turn data tombs into "golden nuggets" of knowledge.

Data mining should be applicable to any kind of information repository. This includes relational databases, data warehouses, transactional databases, advanced database systems, flat files and the WWW. Advanced database system includes Object oriented and object relational databases, and specific application oriented databases, such as spatial databases, text databases and multimedia databases. The challenges and techniques of mining may differ for each of the repository systems.

Relational Databases

A relational database is a collection of tables, each of which is assigned a unique name. Each table consists of a set of attributes (columns or fields) and usually stores a large set of tuples (records or rows). Each tuple in a relational table represents an object identified by a unique key and described by a set of attribute values. A semantic data model, such as an entity-relationship (ER) data model, which models the database as a set of entities and their relationships, is often constructed for relational databases.

Data Warehouses

A data warehouse is a repository of information collected from multiple sources, stored under a unified schema, and which usually resides at a single site. Data warehouses are constructed via a process of data cleaning, data transformation, data integration, data loading, and periodic data refreshing. In order to facilitate decision making, the data in a warehouse are organized around major subjects, such as customer, item, supplier, and activity. The data are stored to

provide information from a historical perspective (such as from the past 5-10 years) and are typically summarized. For example, rather than storing the details of each sales transaction, the data warehouse may store a summary of the transactions per item type for each store or summarized to a higher level, for each sales region.

Transactional Databases

In general, a transactional databases consists of a file where each record represents a transaction. A transaction typically includes a unique transaction identity number (trans_id), and a list of the items making up the transaction (such as items purchased in a store). The transactional database may have additional tables associated with it, which contain other information regarding the salary, such as the data type of the transaction, the customer ID number of the sales person and of the branch at which the sale occurred, and so on.

The transactional database is usually stored in a flat file in a format similar to that of the table in fig 1.1.

sales	
Trans_ID	List of item_IDs
T100	I1,I3,I8,I16
...	...

Fig 1.1 Sales transaction

Advanced Database Systems and Advanced Database Applications

Relational database systems have been widely used in business applications. With the advances of database technology, various kinds of advanced database systems have emerged and are undergoing development to address the requirements of new database applications.

Object-Oriented Databases

Object oriented databases are based on the object oriented programming paradigm, where in general terms, each entity is considered as an object. Data and code relating to an object are encapsulated into a single unit. Each object has associated with it the following:

- A set of variables that describes the objects. These correspond to attributes in the entity-relationship and relational models.
- A set of messages that the object can use to communicate with the other objects or with the rest of the databases system.
- A set of methods, Where each method holds the code to implement a message.
- Upon receiving a message, the method returns a value in response. For instance, the method for the message `get_photo(employee)` will retrieve and return a photo of the given employee object.

Object-Relational Databases

Object-relational databases are constructed based on an object relational data model. This model extends the relational model by providing a rich data type for handling complex objects and object orientation. In addition, special constructs for relational query languages are included to manage the added data types. The object relational model extends the basic relational data model by adding the power to handle complex data types, class hierarchies, and object inheritance as described above. Object-relational databases are becoming increasingly popular in industry and applications.

Spatial Databases

Spatial databases contain spatial related information. Such databases include geographic (map) databases, VLSI chip design databases, and medical and satellite image databases. Spatial data may be represented in raster format, consisting of n-dimensional bit maps or pixel maps. For example, a 2-D satellite image may be

represented as raster data, where each pixel registers the rainfall in a given area. Maps can be represented in a vector format, where roads, bridges, buildings, and lakes are represented as unions of basic geometric constructs, such as points, lines, polygons, and the partitions and networks formed by these shapes.

Geographic databases have a number of applications, ranging from forestry and ecology planning, to providing public service information regarding the location of telephone and electric cables, pipes, and sewage systems. In addition geographic databases are used in vehicle navigation and dispatching systems. An example of such a system for taxis would store a city map with information regarding one-way streets, suggested routes for moving from region A to region B during rush hour, the location of restaurants and hospitals, as well as the current location of each driver.

Temporal Databases and Time-Series Databases

Temporal databases and time-series database both store time related data. A temporal database usually stores relational data that include time-related attributes. These attributes may involve several timestamps, each having different semantics. A time-series database stores sequences of values that change with time, such as data collected regarding the stock exchange.

Text Databases and Multimedia Databases

Text databases are databases that contain word description for objects. These word descriptions are usually not simple keywords but rather long sentences or paragraphs, such as product specifications, error or bug reports, warning messages, summary reports, notes, or other documents. Text databases may be highly unstructured (such as some Web pages on the WWW). Some text databases may be somewhat structured (such as library databases). Text databases with highly regular structures typically can be implemented using relational database systems.

Multimedia databases store image, audio, and video data. They are used in applications such as picture content-based retrieval, voice-mail systems, video-on-demand systems, the WWW and speech based user interfaces that recognize spoken commands. Multimedia databases must support large objects, since data objects such as video can require gigabytes of storage. Specialized storage and search techniques are also required. Since video and audio data require real-time retrieval at a steady and predetermined rate in order to avoid picture or sound gaps and system buffer overflows, such data are referred to as continuous media data.

The World Wide Web

The WWW and its associated distributed information services, such as American Online, Yahoo!, AltaVista, and Prodigy, provide rich, world-wide, on-line information services, where data objects are linked together to facilitate interactive access. Users seeking information provide ample opportunities and challenges for data mining. For example, understanding user access patterns will not only help improve system design (by providing efficient access between highly correlated objects), but also leads to better marketing decisions (e.g., by placing advertisements in frequently visited documents, or by providing better customer/user classification and behavior analysis). Capturing user access patterns in such distributed information environments is called mining path traversal patterns.

1.2 Data Mining Functionality- What kind of patterns can be mined

Data mining functionality are used to specify the kind of patterns to be found in data mining tasks. In general , data mining tasks can be classified into two categories: Descriptive and Predictive. Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining tasks perform inferences on the current data in order to make predictions.

Concept / Class Description: Characterization and Discrimination

Data Characterization is a summarization of the general characteristics or features of a target class(data of the class under study) of data. The data corresponding to the user-specified class are typically collected by a database query. For example, to study the characteristics of software products whose sales increased by 10 % in the last year, the data related to such products can be collected by executing an SQL query.

Data discrimination is a comparison of the general features of target class data objects with the general features of objects from one or a set of contrasting classes. The target and contrasting classes can be specified by the user, and the corresponding data objects retrieved through database queries. For e;g the user may like to compare the general features of software products whose sales increased by 10 % in the last year with those whose sales decreased by at least 30 % during the same period. The methods used for data discrimination are similar to those used for data characterization.

Association Analysis

Association analysis is the discovery of association rules showing attribute-value conditions that occur frequently together in a given set of data. Association analysis is widely used for market basket or transaction data analysis. [2][3][5]

More formally, association rules are of the form $X \Rightarrow Y$, that is, " $A_1 \wedge \dots \wedge A_m \rightarrow B_1 \wedge \dots \wedge B_n$ ", where A_i (for $i \in \{1, \dots, m\}$) and B_j (for $j \in \{1, \dots, n\}$) are attribute-value pairs. The association rule $X \Rightarrow Y$ is interpreted as "database tuples that satisfy the conditions in X are also likely to satisfy the condition in Y".

Suppose , as a marketing manager of ALLElectronics , you would like to determine which items are frequently purchased together within the

same transactions. An example of such a rule is
 $\text{contains}(T, \text{"computer"}) \Rightarrow \text{contains}(T, \text{"software"})$
[support = 1%, confidence = 50 %]
meaning that if a transaction, T, contains "computer", there is a 50% chance that it contains "software" as well, and 1% of all the transactions contains both. This association rule involves a single attribute or predicate (i.e., contains) that repeats. Association rules that contain a single predicate are referred to as single dimensional association rules. Dropping the predicate notation, the above rule can be written simply as "computer \Rightarrow software[1%, 50%].

Classification and Prediction

Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of training data (i.e., data objects whose class label is known).

Classification can be used for predicting the class label of data objects. However in many applications, users may wish to predict some missing or unavailable data values rather than class labels. This is usually the case when the predicted values are numerical data and is often specifically referred to as prediction. Although prediction may refer to both data value prediction and class label prediction, it is usually confined to data value prediction and thus is distinct from classification. Prediction also encompasses the identification of distributed trends based on the available data.

Cluster Analysis

Unlike classification and prediction, which analyze class-labeled data objects, clustering analyzes data objects without consulting a known class label. In general, the class labels are not present in the training data simply because they are not known to begin with. Clustering can be used to generate such labels. The objects are clustered or grouped

based on the principal of maximizing the intraclass similarity and minimizing the interclass similarity. That is , clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate taxonomy formation, that is, the organization of observations into a hierarchy of classes that group similar events together.

Outlier Analysis

A database may contain data objects that do not comply with the general behavior or model of the data. These data objects are outliers. Most data mining methods discard outliers as noise or exceptions. However, in some applications such as fraud detection, the rare events can be more interesting than the more regularly occurring ones. The analysis of outlier data is referred to as outlier mining. For e.g., outlier analysis may uncover fraudulent usage of credit cards by detecting purchases of extremely large amounts for a given account number in comparison to regular charges incurred by the same account.

CHAPTER 2

HIERARCHICAL CENSORED PRODUCTION RULES / HIERARCHICAL PRODUCTION RULES

2.1 Introduction

To an ordinary logic based reasoning system you cannot tell much about the way you want it to perform its task, for example you cannot give the following instructions:

- Give me a reasonable answer immediately even if it is somewhat general and if there is enough time then give me a more specific answer.
- Give me a reasonable answer immediately. If there is enough time then tell me you are more confident in the answer or change your mind and give me another better answer
- Give me a reasonable answer immediately even if it is somewhat less certain and if you have enough time then give me a more specific answer.[7]

A system having represented real word knowledge should also be capable of handling these types of requirements for natural and efficient, reasoning. In the real world both human and computer often has to reason using insufficient, incomplete or tentative premises. Moreover both are subject to constraints of time and memory. Variable precision logic is concerned with problems of reasoning with incomplete information and constraints on resources. It offers mechanism for handling trade-off between the precision of inferences and computational efficiency of deriving them. The certainty and specificity are the two aspect of precision. Following Michalski and Winston , a system that gives more specific answer given more time is called 'variable specific system'. A system that gives more certain answer given more time is called 'variable certainty system'. There can be various combination of the two systems, reflecting the fact specificity and certainty are inversely related. We can gain specificity at

the expense of certainty, or vice-versa. Consider a query 'What is John doing?' given that it is Sunday, the quickest possible answer to this query may be that 'probably John is working in the yard'. A more certain answer taking into account the nice weather is that. Certainly John is working in the yard rather reading.

Michalski and Winston have suggested the censored Production Rule as an underlying representational and computational mechanism to enable logic-based systems to exhibit variable precision in which certainty varies while specificity remains constant. The extension of CPR is Hierarchical Censor Production Rules system of knowledge representation which exhibits both the variable certainty as well as the variable specificity .

At any state of the reasoning system, it is valuable for a reasoning process to have the information about a set of applicable rules because then it need not find them using exhaustive search of the whole rule-base. For example consider the query 'what is John doing?' issued to the reasoning system. Having found by the reasoning process that 'John is working in the yard'. The next line of action taken might be to get a more specific answer if there are enough resources available. At this state of the reasoning process the next set of rules selected by an intelligent reasoning system should give decision of the following type:

- John is raking leaves
- John is watering
- John is shaping plants

Rather than the rules which give decision of the type:

- John is eating fruits
- John is reading a story book
- John is watching a movie

These later decisions are totally unrelated to the previous inferred decision, i.e, 'John is working in the yard'. A reasoning system should avoid a trial of out of

context rules, because it might request some irrelevant information which the user might not like. Intelligent systems should also be able to quickly discard most of the task irrelevant information and rather concentrate on main line of reasoning. To certain extent the HCPRs system will be able to incorporate the above requirements on the reasoning system.

2.2 Hierarchical Censored Production Rules

A production Rule has the following form

<If precondition Then action>

Where precondition is left part of the rule sometimes called antecedent which when satisfied leads to take the action denoted by right side of the rule sometimes called consequent. A production rules system can capture much of the simple human problem solving capability effectively. However not all human problem solving methods are easily representable in the production rules system, because

- (i) For each rule information has to exist in the system somewhere as to its context of use. This can result in overlarge rule antecedents or in implicit knowledge such as that contained in rule order. Either way control knowledge is often not clear.
- (ii) Sets of rules in production rules system have no intrinsic structure which makes maintains of large knowledge base difficult.
- (iii) The matching involved in the match select-fire is an inherently inefficient process.
- (iv) The relatively simple syntax of a production rule is unable to capture the inherent uncertainty present in the real world knowledge.

To capture the uncertain and imprecise knowledge about the real world Michalski and Winston have introduced the concept of Censor Production Rule (CPR). A CPR is a production rule augmented with exception conditions to the rule. A CPR has the following syntax:

<If precondition Then action Unless censor : γ, δ >

i.e., $B \Rightarrow A \lfloor C : \gamma, \delta$

where 'B' is antecedent part, 'A' is consequent and 'C' is censor part of the rule. Symbols ' γ ' and ' δ ' represent the 0-level and 1-level strength of implication respectively CPR is unable to capture the structure inherent in the knowledge about real world and hence would not impart control over the specificity part of precision in decision making.

A HCPR is a CPR augmented with specificity information which can be made to exhibit variable precision in the reasoning such that both certainty of belief in a conclusion and its specificity may be controlled by the reasoning process.

A HCPR has the following form:

<Decision (If precondition)
(**Unless** censor_conditions)
(**Generality** general_information)
(**Specificity** specific_information)
: γ, δ >

i.e., $A(-B)(\lfloor C)(G\% G)(\$ S) : \gamma, \delta$

Where the symbols ':-', ' \lfloor ', 'G %', and '\$' are used for 'IF', 'Unless', 'Generality' and 'Specificity' operators respectively and symbols 'B', 'C', 'G', and 'S' denote the corresponding information relegated with them.

From a control viewpoint HCPRs are intended for situations in which the condition 'A (-B)' holds frequently and the assertion 'C' holds rarely. Systems using HCPRs are free to ignore the exception conditions when resources are tight. Given more time, the exception conditions are examined lending credibility to high-speed answer or changing them. From a logical viewpoint the 'Unless' operator between 'A' and 'C' acts as the 'xor' operator. From an expository viewpoint the A(-B)' part of a HCPR expresses important information while the ' $\lfloor C$ ' part acts only as a switch that changes the polarity of 'A' to 'Not A' when 'C' holds.

The specificity information 'S' in a HCPR is the clue about the next set of more specific concepts (goals, decision, consequents or actions) in a knowledge-base which are the most relevant and which are the most likely to be satisfied after successful execution of that HCPR. Under backward chaining of reasoning the information relegated with 'specificity' operator may not be so useful, and hence may not be used by the reasoning process.

The general information 'G' in a HCPR , is the clue about the next general concept related to the concept 'A' in hierarchy. Under forward chaining of reasoning the information relegated with 'Generality ' operator could be neglected by the reasoning process completely.

2.3 HCPR-tree

A HCPR-tree is a collective and systematical representation of all the related HCPRs about a given problem domain. Here 'collective' is the set of all rules or definitions related to a most generalized concept in the rule-base. A HCPR in the HCPR-tree is the smallest and simplest chunk of knowledge that may be created, modified, or removed without directly affecting the other HCPRs in the HCPR-tree (because of its declarative nature of representation).

The general concepts in a HCPR-tree are represented at relatively lower level of specificity and the specific concepts are represented at relatively higher level of specificity. Thus a HCPR-tree is a systematical representation of all the HCPRs for related problem domain.

In a HCPR-tree all the sub concepts of an immediate general concepts will be called siblings. These sibling concepts are assumed to be related to each other by mutually exclusive property and this relationship would be made explicit by using the logical operator 'xor' between the siblings in the specificity information part of the rule.

Consider a particular example of a HCPR-tree as shown in fig 2.1 ,for daily life queries which represent

A plan to find answers to the queries of the type 'What is X doing?' when supplied with relevant input data. The rule-base for the HCPR-tree might be represented as follows:

/* level 0 */

```
Is_in_city(X,Y)
    :-
    Lives_in_city(X,Y)
    |
    Is_on_tour(X)
    $
    Is_at_home(X) xor Is_outside_home(X)
```

/*level 1*/

```
Is_at_home(X)
    :-
    Time(night)
    |
    Is_doing_overtime(X) or Works_in_night_shift(X)
    G%
```

```
Is_in_city(X,Y)
```

```
Is_outside_home(X)
```

```
    :-
    Time(day)
    |
    Is_ill(X) or Is_atching_world_cup(X) or Bad_weather or
    Disturbances_in_city
    G%
```

```

Is_in_city(X,Y)
  $
  Is_working_outdoor xor Is_entertaining_outdoor(X)

```

/*level 2*/

```

Is_working_outdoor(X)
  :-
  Day(working)
  |
  National_holiday or Is_unemployed(X)
  G%
  Is_outside_home(X)

Is_entertaining_outdoor(X)
  :-
  Day(Sunday)
  |
  Met_an_accident(X)
  G%
  Is_outside_home(X)

```

To see how a system based on HCPRs system of representation concentrate on main line of reasoning consider the above HCPR-tree for answering queries of the type 'What is X doing?'. The reasoning system would first ask the query 'Which city does person X lives in?', from the user. Having got the reply that 'X lives in city Y'(variable 'Y' would be initialized to the name of the city) the system may conclude that 'X is in the city Y'. In order to strengthen this conclusion further the reasoning system may try to find 'Is X on tour?'. If it is found to be true then the system would stop with the conclusion,'X is on tour'. Otherwise this censor condition found false then system may proceed to get a more specific

answer of the type 'X is at home' or 'X is outside home' by asking or checking whether it is night or day time. Similarly depending upon the previously inferred general conclusion, say, 'X is outside home', the system could attempt for a more specific decision like 'X is working outdoor' or 'X is entertaining outdoor'.

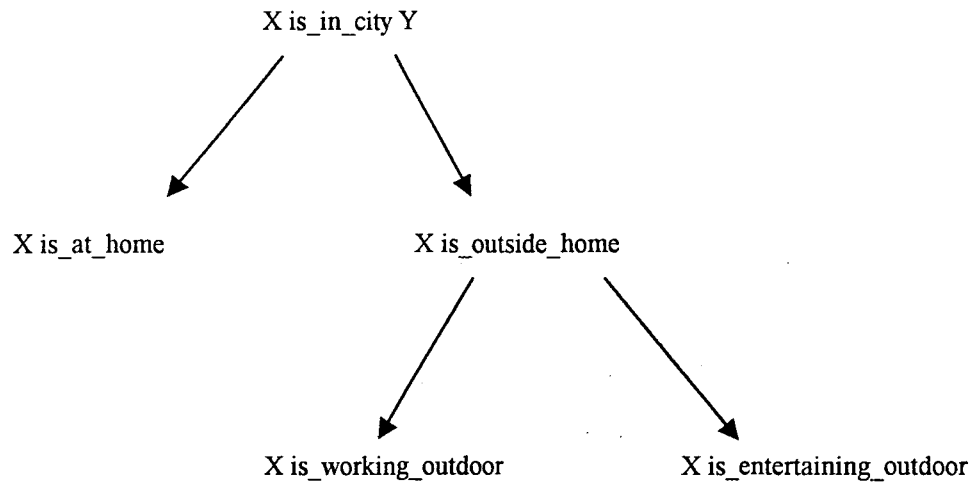


Fig 2.1 A HCPR-tree for daily life queries.

2.4 Inference Rules

The information relegated with the previous operators (i.e., 'If, Unless, Specificity, generality) in a HCPR is optional, and hence some or all of these information may be incomplete or even totally absent.

Consider a HCPRs system in which censor condition is totally absent in the HCPR and which has only two levels of certainty factor (0 for false and 1 for true). A rule in such a HCPRs system may be called Hierarchical Production Rule (HPR) which is a HCPR without 'Unless ' operator.

As an example the HCPR-tree "R" in fig 2.2 showing the general concept of 'Excitement' may be represented in a rule-base as follows.

R:

- Excitement : - state of agitation
\$ (Distress xor Delight)
- Distress : - extreme pain
G% Excitement
\$ (Fear xor Shame xor Anger)
- Delight :- great pleasure
G% Excitement
\$ (Affection xor Joy xor Elation)
- Fear : - danger
G% Distress
- Shame : - guilt feeling
G% Distress
- Anger : - real or fancied injury
G% Distress
- Affection :- Kindness or love
G% Delight
- Joy : - gladness
G% Delight
- Elation : - pride from success
G% Delight

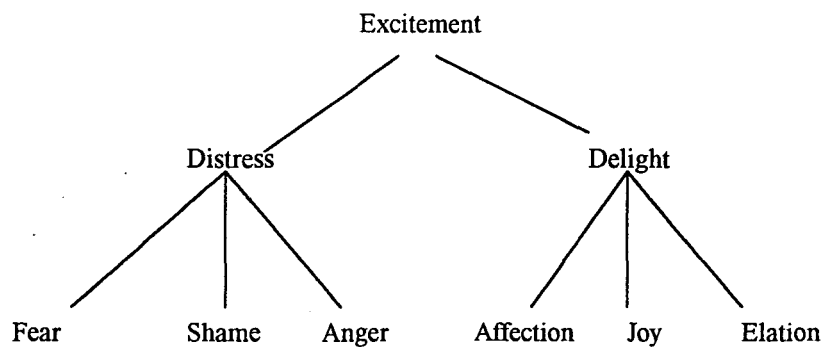


Fig 2.2 A HCPR-tree to represent different feelings of a person.

The rule–base for 'R' may be used to find an appropriate word required to describe the feeling of a person. In this system only most specialized knowledge about an object or action is required to be observed and stored in the fact-base whereas the generalized facts about the same object or action may be inferred using the fact-base and the rule-base about the world.

For example, using (i) the facts that 'John is in the state of agitation ' and 'John is having extreme pain' in the fact base and (ii) the HCPR-tree 'R' in the rule base a reasoning process will infer that 'John is distressed ' and 'John is excited'. Similarly using the single fact that 'Mary is in joy' in the fact-base it will infer that 'Mary is excited' and 'Mary is delighted'. Also using the same fact in the fact-base the queries of the type 'Did Mary feel distressed?'. 'Did Mary feel ashamed?', "Did Mary feel elated?", etc., could be answered in the negative.

CHAPTER 3

AUTOMATIC GENERATION OF HIERARCHICAL STRUCTURES

3.1 Introduction

Concept hierarchies organize data and concepts in hierarchical forms or in certain partial order, which helps expressing knowledge and data relationships in databases in concise, high level terms and thus plays an important role in knowledge discovery processes. Concept hierarchies could be provided by knowledge engineers, domain experts, users or embedded in some data relations, However, it is sometimes desirable to automatically generate some concept hierarchies or adjust some given hierarchies for particular learning tasks.

With the rapid growth in size and number of available databases in commercial, industrial, administrative and other applications ,it is necessary and interesting to examine how to extract knowledge automatically from huge amounts of data. By extraction of knowledge in databases, large databases will serve as a rich, reliable source for knowledge generation and verification and the discovered knowledge can be applied to information management, query processing, decision making, process control and many other applications.

Therefore, knowledge discovery in databases or data mining has been considered as one of the most important research topics in 1990s by both machine learning and database researchers.[1][6]

3.2 Concept Hierarchy And Its Role In Knowledge Discovery In Databases

A concept hierarchy defines a sequence of mappings from a set of lower-level concepts to their higher-level correspondences. Such mappings may organize the set of concepts in partial order, such as in the shape of a tree (a hierarchy, a taxonomy), a lattice, a directed acyclic graph, etc., although they are still called "hierarchies" for convenience.

A concept hierarchy can be defined on one or a set of attribute domains. Suppose a hierarchy H is defined on a set of domains D_1, \dots, D_k , in which different levels of concepts are organized into a hierarchy. The concept hierarchy is usually partially ordered according to a general-to-specific ordering. The most general concept is the null description (described by a reserved word "ANY"); whereas the most specific concepts correspond to the specific values of attributes in the database.

Formally, we have $H_i : D_1 \times \dots \times D_k \Rightarrow H_{i-1} \Rightarrow \dots \Rightarrow H_0$, where H_i represents the set of concepts at the primitive level, H_{i-1} represents the concepts at one level higher than those at H_i , etc., and H_0 , the highest level hierarchy, may contain solely the most general concept, "ANY". Since concept hierarchies define mapping rules between different levels of concepts, they are in general data or application specific. Many concept hierarchies, such as birthplace(city, province, country), are actually stored implicitly in the database, such as in different attributes or different relations, which can be made explicit by specifying certain attribute mapping rules. Moreover, some concept mapping rules can be specified by deduction rules or methods (such as in object oriented databases) and be derived by deduction or computation. For example, the floor area of a house can be computed from the dimensions of each segment in the house by a spatial computation algorithm, and then mapped to a high level concept, such as small, large, etc. defined by deduction rules.

TH - 11279



The mappings of a concept hierarchy or a portion of it may also be provided explicitly by a knowledge engineer or a domain expert. For example “status: { freshman, sophomore, junior, senior} → undergraduate”, “annual income: {1,000, ..., 25,000} → low income” , etc., can be specified by domain experts.

This is often reasonable even for large databases since a concept hierarchy registers only the distinct discrete attribute values or ranges of numerical values for an attribute which are, in general not very large and can be specified by domain experts. Furthermore, some concept hierarchies can be discovered automatically.

Different concept hierarchies can be constructed on the same attribute(s) based on different viewpoints or preferences. For example, the birthplace could be organized according to administrative regions, geographic locations, size of cities, etc. Usually a commonly referenced concept hierarchy is associated with an attribute as the default one. Other hierarchies can be chosen explicitly by preferred users in a learning process. Also, it is sometimes preferable to perform induction in parallel along more than one concept hierarchy and determine an appropriate representation based on later generalization results.

3.3 Concept Hierarchy Generation

Discretization techniques can be used to reduce the number of values for a given continuous attribute, by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values . Reducing the number of values for an attribute is especially beneficial if decision-tree-based methods of classification mining are to be applied to the preprocessed data. These methods are typically recursive, where a large amount of time is spent on sorting the data at each step. Hence , the smaller the number of distinct values to sort, the faster these methods should be. Many discretization techniques can be applied

recursively in order to provide a hierarchical or multiresolution partitioning of the attribute values, known as concept hierarchy.

A concept hierarchy for a given numeric attribute defines a discretization of the attribute. Concept hierarchies can be used to reduce the data by collecting and replacing low-level concepts (such as numeric values for the attribute age) by higher level concepts (such as young, middle-aged, or senior). Although details is lost by such data generalization, the generalized data may be more meaningful and easier to interpret, and will require less space than the original data. Mining on a reduced data set will require fewer input/output operations and be more efficient than mining on a large, ungeneralized data set.

3.3.1 Generation Of Concept Hierarchy For Numerical Data

It is difficult and laborious to specify concept hierarchies for numeric attributes due to the wide diversity of possible data ranges and the frequent updates of data values. Such manual specification can also be quite arbitrary.

Concept hierarchies for numeric attributes can be constructed automatically based on data distribution analysis. Five methods for numeric concept hierarchy generation are discussed: binning, histogram analysis, cluster analysis, entropy-based discretization and data segmentation by "natural partitioning".

Binning

Binning methods smooth a sorted data value by consulting its "neighborhood", that is, the values around it. The sorted values are distributed into a number of "buckets", or "bins". Because binning methods consult the neighborhood of values, they perform local smoothing. fig 3.1 illustrates some binning techniques. In this example, the data for price are first sorted and then partitioned into equidepth bins of depth 3 (i.e., each bin contains three values). In smoothing by bin means, each value in a bin is replaced by the mean value of the bin. For example, the mean values 4, 8 and 15 in Bin 1 is 9. Therefore, each

original value in this bin is replaced by the value 9. Similarly, smoothing by bin medians can be employed, in which each bin value is replaced by the bin median. In smoothing by bin boundaries, the minimum and maximum values in a given bin are identified as the bin boundaries. Each bin value is then replaced by the closest boundary value. In general, the larger the width, the greater the effect of the smoothing. Alternatively, bins may be equiwidth, where the interval range of values in each bin is constant.

Sorted data for price (in dollars): 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equidepth) bins:

Bin 1 : 4, 8, 15

Bin 2 : 21, 21, 24

Bin 3 : 25, 28, 34

Smoothing by bin means:

Bin 1 : 9, 9, 9

Bin 2 : 22, 22, 22

Bin 3 : 29, 29, 29

Smoothing by bin boundaries:

Bin 1 : 4, 4, 15

Bin 2 : 21, 21, 24

Bin 3 : 25, 25, 34

Fig 3.1 Binning methods for data smoothing

Histogram Analysis

Histogram can also be used for discretization. Fig 3.2 presents a histogram showing the data distribution of the attribute price for a given data set. For example, the most frequent price range is roughly \$300-\$325. Partitioning rules

can be used to define the ranges of values. For instance, in an equiwidth histogram, the values are partitioned into equal-sized portions or ranges (eg., (\$0 . . . \$100],(\$100...\$200],...,\$(900...\$1000]). With an equiwidth histogram, the values are partitioned so that, ideally, each partition contains the same number of data samples. The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A minimum interval size can also be used per level to control the recursive procedure. This specifies the minimum width of a partition, or the minimum number of values for each partition at each level.

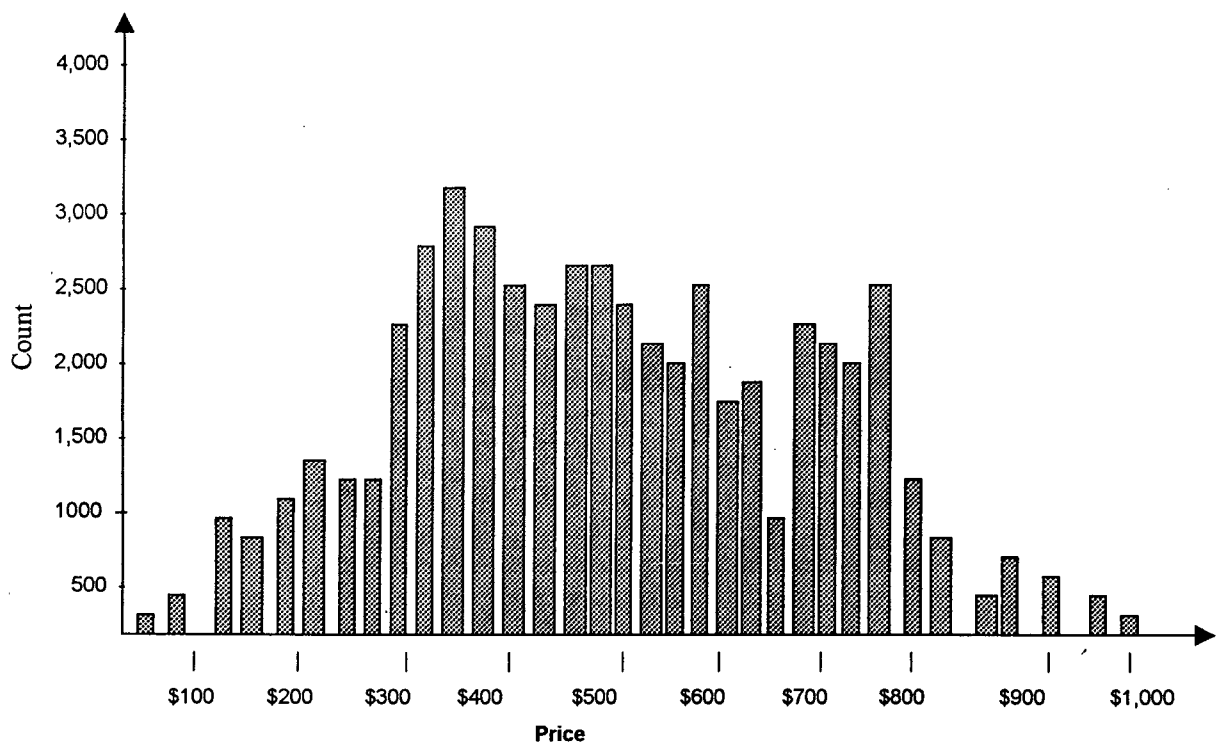


Fig 3.2 Histogram showing the distribution of values for the attribute price

Cluster Analysis

A clustering algorithm can be applied to partition data into clusters or groups. Each cluster forms a node of a concept hierarchy, Where all nodes are at the same concept level. Each cluster can be further decomposed into several sub clusters, forming a lower level of the hierarchy. Clusters may also be grouped together in order to form a higher conceptual level of the hierarchy.

Entropy based Discretization

An information-based measure called entropy can be used to recursively partition the values of a numeric attribute A , resulting in a hierarchical discretization. Such a discretization forms a numerical concept hierarchy for the attributes. Given a set of data tuples, S , the basic method for entropy-based discretization of A is as follows:

Each value of A can be considered a potential interval boundary or threshold T . For Example, a value v of A can partition the samples in S into two subsets satisfying the conditions $A < v$ and $A \geq v$, respectively, thereby creating a binary discretization.

Given S , the threshold value selected is the one that maximizes the information gain resulting from the subsequent partitioning. The information gain is

$$I(S, T) = |S_1/S| \text{Ent}(S_1) + |S_2/S| \text{Ent}(S_2),$$

Where S_1 and S_2 correspond to the samples in S satisfying the conditions $A < T$ and $A \geq T$, Respectively. The entropy function Ent for a given set is calculated based on the class distribution of the samples in the set. For example, given m classes, the entropy of S_1 is

$$\text{Ent}(S_1) = - \sum p_i \log_2(p_i),$$

Where p_i is the probability of class i in S_1 , determined by dividing the number of samples of class i in S_1 by the total number of samples in S_1 . The value of $\text{Ent}(S_2)$ can be computed similarly.

The process of determining a threshold value is recursively applied to each partitioned obtained, until some stopping criterion is met, such as

$$\text{Ent}(S) - I(S,T) > \delta.$$

Entropy-based discretization can reduce data size. Unlike the other methods mentioned here so far, entropy-based discretization uses class information. This makes it more likely that the interval boundaries are defined to occur in places that may help improve classification accuracy. The information gain and entropy measures described here are also used for decision tree induction.

Segmentation by Natural Partitioning

Although binning, histogram analysis, clustering, and entropy-based discretization are useful in the generation of numerical hierarchies, many users would like to see numerical ranges partitioned into relatively uniform, easy-to-read intervals that appear intuitive or "natural." For example, annual salaries broken into ranges like (\$50,000, \$60,000) are often more desirable than ranges like (\$51,263.98, \$60,872.34), obtained by some sophisticated clustering analysis.

The 3-4-5 rule can be used to segment numeric data into relatively uniform, "natural" intervals. In general, the rule partitions a given range of data into 3, 4, or 5 relatively equiwidth intervals, recursively and level by level, based on the value range at the most significant digit. The rule is as follows:

- If an interval covers 3,6,7, or 9 distinct values at the most significant digit, then partitioned the range into 3 intervals (3 equiwidth intervals for 3,6,9, and 3 intervals in the grouping of 2-3-2 for 7).
- If it covers 2,4,8 distinct values at the most significant digit, then partition the range into 4 equiwidth intervals.
- If it covers 1,5, or 10 distinct values at the most significant digit, then partition the range into 5 equiwidth intervals.

The rule can be recursively applied to each interval, creating a concept hierarchy for the given numeric attribute. Since there could be some dramatically large positive or negative values in a data set, the top-level segmentation, based merely on the minimum and maximum values, may derive distorted results. For example the assets of a few people could be several orders of magnitude higher than those of others in a data set. Segmentation based on the maximal asset values may lead to a highly biased hierarchy. Thus the top-level segmentation can be performed based on the range of data values representing the majority(e.g., 5th percentile to 95th percentile) of the given data . The extremely high or low values beyond the top-level segmentation will form distinct interval(s) that can be handled separately, but in a similar manner.

3.3.2 Concept Hierarchy Generation for Categorical Data

Categorical data are discrete data. Categorical attribute have a finite (but possibly large) number of distinct values, with no ordering among the values. Examples include geographic location, job category, and item type. There are several methods for the Generation of concept hierarchy for categorical data.

Specification of a partial ordering of attributes explicitly at the schema level by users or experts:

Concept hierarchies for categorical attributes or dimension typically involve group of attributes user or a group can easily define a concept hierarchy by specifying

a partial or total ordering of the attributes at the schema level. For example, a relational database or a dimension location of a data warehouse may contain the following group of attributes : street, city , province_or_state, country. A hierarchy can be defined by specifying the total ordering among these attributes at the schema level , such as street < city < province_or_state < country.

Specification of a portion of a hierarchy by explicit data grouping:

This is essentially the manual definition of a portion of a concept hierarchy. In a large database, it is unrealistic to define an entire concept hierarchy by explicit value enumeration. However it is realistic to specify explicit grouping for a small portion of intermediate-level data. For example, after specifying that province and country form a hierarchy at the schema level, one may like to add some intermediate levels manually, such as defining explicitly "{Alberta,Saskatchewan,Manitoba} ⊂ prairies_Canada" and "{British Columbia, prairies_Canada} ⊂ western_Canada".

Specification of a set of attributes, but not of their partial ordering:

A user specify a set of attributes forming a concept hierarchy, but omit to explicit state their partial ordering. The system can then try to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy.

Example 3.1

Suppose a user selects a set of attributes, street, country, province_or_state, and city, for a dimension location from the database AllElectronics, but doesn't specify the hierarchical ordering among the attributes.

The concept hierarchy for location can be generated automatically as follows. First, sort the attributes in ascending order based on the number of distinct

values in each attribute. This results in the following (where the number of distinct values per attribute is shown in parentheses) "country (15), province_or_state(365),city(3567),and street (9674,339). Second , generate the hierarchy from the top down according to the sorted order, with the first attribute at the top level and the last attribute at the bottom level. The resulting hierarchy is shown in fig 3.3, finally the user can examine the generated hierarchy, and when necessary, modify it to reflect desired semantics relationships among the attributes. In this example, it is obvious that there is no need to modify the generated hierarchy.

Note that this heuristic rule cannot be pushed to the extreme since there are obvious cases that do not follow such a heuristic. For example, a time dimension in a database may contain 20 distinct years, 12 distinct months, and 7 distinct days of the week. However, this doesn't suggest that the time hierarchy should be "year<month<days_of the week", with days_of_the_week at the top of the hierarchy.

Specification of only a partial set of attributes: Sometimes a user can be sloppy when defining a hierarchy, or may have only a vague idea about what should be included in a hierarchy. Consequently, the user may have included only a small subset of the relevant attributes in a hierarchy specification. For example, instead of including all the hierarchically relevant attributes for location, the user may have specified only street and city. To handle such partially specified hierarchies, it is important to embed data semantics in the database schema so those attributes with tight semantic connection can be pinned together. In this way, the specification of one attribute may trigger a whole group of semantically tightly linked attributes to be "dragged in" to form a complete hierarchy. Users, however, should have the option to override this feature, as necessary.

Example 3.2

Suppose that a database system has pinned together the five attributes number, street, city, province_or_state and country because they are closely linked semantically, regarding the notion of location. If a user were to specify only the attribute City for a hierarchy defining location, the system may automatically drag in all of the above five semantically related attributes to form a hierarchy. The user may choose to drop any of these attributes, such as number and street, from the hierarchy, keeping city as the lowest conceptual level in the hierarchy.

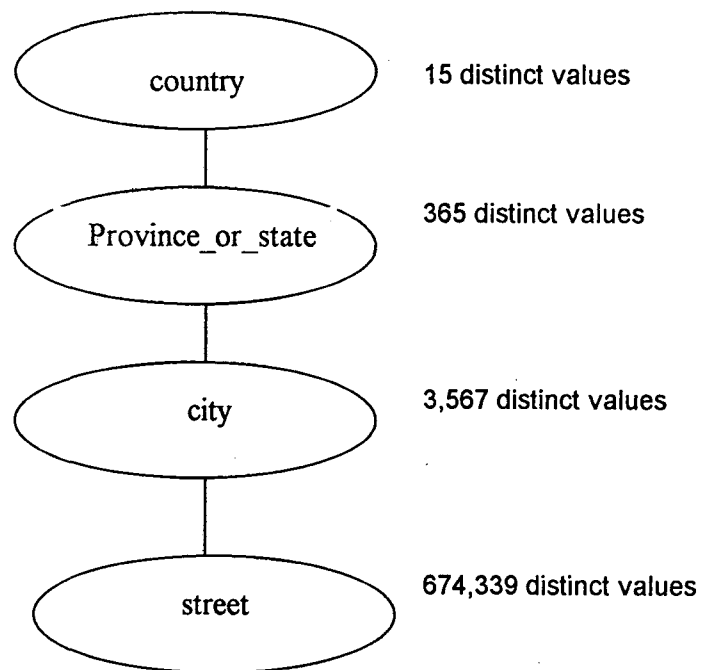


Fig 3.3 Automatic generation of a schema concept hierarchy based on the number of distinct attribute values

3.4 Algorithm (Generation Of HPRs)

Algorithm 3.1

Input : Given a table 'T' with tuples and attributes denoting decision and premises.

Output : HPR Tree

Step 1 : tag=1, R=NULL

Step 2 : Find all unmark tuples in the table 'T' with minimum count and place in the set D_i

Step 3 : if ($D_i = \phi$) then GOTO Step 7

Step 4 : if (tag==1) then GOTO Step 6

Step 5 : In set D_i , find tuples (elements) with common premises (sub-elements) say p

make p as root and participating element of set D_i as children

if(non participating element in set D_i)

{

Make R=root node with p and nonparticipating element as children;

}

else R=p;

mark elements of set D_i ,tag=0

GOTO Step 7

Step 6 : Make elements of set D_i children of R. Mark elements of set D_i

Step 7 : For each element t_j in set D_i . Find unmark element that have t_j as subset, say D_T set.

Replace T with D_T set.

Step 8 : If ($j \leq$ number of elements in D_i) then $R=t_j$, GOTO Step2
else GOTO Step 9

Step 9 : if any unmark tuple left GOTO step 2

step10: stop

The algorithm is explained below

Example 3.3

Consider the following production rules

1. D1 IF P1 P2 P3
2. D2 IF P1 P8 P9 P14
3. D3 IF P1 P2
4. D4 IF P1 P5 P8 P9
5. D5 IF P1 P2 P6
6. D6 IF P1 P4
7. D7 IF P1 P2 P6 P7
8. D8 IF P1 P8 P9
9. D9 IF P1 P8 P9 P12 P13 P14
10. D10 IF P1
11. D11 IF P1 P8 P9 P11 P14
12. D12 IF P1 P2 P6 P10
13. D13 IF P1 P15

where D_1 is Decision and P_i are premises.

Following is the representation of production rules in the form of Table 3.1 .In the Table, Tuples and Attributes represents decisions and premises respectively .For example rule 1. with decision D_1 has three premises P_1, P_2 and P_3 . These 3 premises are mark in the first tuple, that corresponds to decision D_1

Table 3.1

Count	Decision	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
3	D1	√	√	√												
4	D2	√							√	√					√	
2	D3	√	√													
4	D4	√				√			√	√						
3	D5	√	√				√									
2	D6	√			√											
4	D7	√	√				√	√								
3	D8	√							√	√						
6	D9	√							√	√			√	√	√	
1	D10	√														
5	D11	√							√	√		√			√	
4	D12	√	√				√				√					
2	D13	√														√

following is the execution algorithm

Step 1 : tag=1, R=NULL

Step 2 : Di={D10}

Step 3 : Condition is false GOTO Step 4

Step 4 : tag==1, GOTO Step 5

Step 5 : Di contains one element so make it root (p=root), there is no non-participating element hence assign R=p; mark D10, tag=0, GOTO Step 7

Step 7 : for t1=D10 do,

T=D-T={D1,D2,D3,D4,D5,D6,D7,D8,D9,D11,D12,D13}

Step 8 : R=t1, GOTO Step 2

Following tree is Created After above steps



In the Table D10 tuple is marked.

2nd Iteration

Step 2 : Di={D3,D6,D13}

Step 3 : Condition is false GOTO Step 4

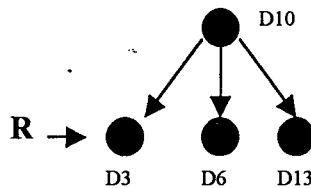
Step 4 : Condition is false GOTO Step 6

Step 6 : Make D3,D6 and D13 children of D10. Mark tuples D3,D6 and D13

Step 7 : for $t_j = D3, D6$ and $D13$. take $t_1 = D3$, $T = D - T = \{D1, D5, D7, D12\}$

Step 8 : $R = t_1$, GOTO Step 2

Following tree is obtained after 2nd Iteration



The status of original Table is as follows where tuples D10, D3, D6 and D13 are marked.

Table 3.2

Count	Decision	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
3	D1	√	√	√												
4	D2	√							√	√					√	
2	√D3	√	√													
4	D4	√				√			√	√						
3	D5	√	√				√									
2	√D6	√			√											
4	D7	√	√				√	√								
3	D8	√							√	√						
6	D9	√							√	√			√	√	√	
1	√D10	√														
5	D11	√							√	√		√			√	
4	D12	√	√				√				√					
2	√D13	√														√

Iteration 3rd

Step 2 : $D_i = \{D1, D5\}$

Step 3 : condition is false GOTO Step 4

Step 4 : condition is false GOTO Step 6

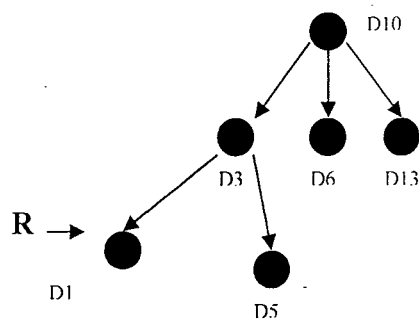
Step 6 : make D1 and D5 children of D3. Mark D1 and D5

Step 7 : for $t_j = D1$ and $D5$ take $t_1 = D1$, $T = D - T = \{\phi\}$

Step 8 : $R = t_1$, GOTO Step 2

In Step 7 D-T is empty as there is no element (tuple) in the table that has D1 as subset.

Following tree is obtained after 3rd Iteration



The status of original Table is as follows where tuples D10,D3,D6, D13, D1 and D5 are marked.

Table 3.3

Count	Decision	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
3	√D1	√	√	√												
4	D2	√							√	√					√	
2	√D3	√	√													
4	D4	√				√			√	√						
3	√D5	√	√				√									
2	√D6	√			√											
4	D7	√	√				√	√								
3	D8	√							√	√						
6	D9	√							√	√			√	√	√	
1	√D10	√														
5	D11	√							√	√		√			√	
4	D12	√	√				√				√					
2	√D13	√														√

Iteration 4th

Step 2 : $D_i = \{0\}$

Step 3 : condition is true GOTO Step 7

Step 7 : for $t_j = D1$ and $D5$ take $t2 = D5$, $T = D - T = \{D7, D12\}$

Step 8 : $R = t2$, GOTO Step 2

Iteration 5th

Step 2 : $D_i = \{D7, D12\}$

Step 3 : condition is false GOTO Step 4

Step 4 : condition is false GOTO Step 6

Step 6 : make D7 and D12 children of D5. Mark D7 and D12

Step 7 : for $t_j = D7$ and $D12$ take $t_1 = D7$, $D-T = \{\phi\}$

Step 8 : $R = t_1$, GOTO Step 2

In Step 7 D-T is empty as there is no element (tuple) in the table that has D7 as subset.

Iteration 6th

Step 2 : $D_i = \{0\}$

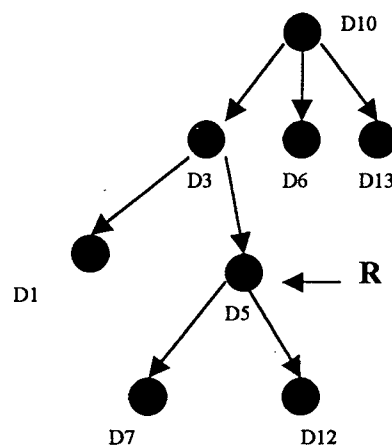
Step 3 : condition is true GOTO Step 7

Step 7 : for $t_j = D7$ and $D12$ take $t_2 = D12$, $D-T = \{\phi\}$

Step 8 : $R = t_2$, GOTO Step 2

In Step 7 D-T is empty as there is no element (tuple) in the table that has D12 as subset.

Following tree is obtained after 6th Iteration



The status of original Table is as follows where tuples D10,D3,D6,D13,D1,D5,D7 and D12 are marked.

Table 3.4

Count	Decision	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
3	√D1	√	√	√												
4	D2	√							√	√					√	
2	√D3	√	√													
4	D4	√				√			√	√						
3	√D5	√	√				√									
2	√D6	√			√											
4	√D7	√	√				√	√								
3	D8	√							√	√						
6	D9	√							√	√			√	√	√	
1	√D10	√														
5	D11	√							√	√		√			√	
4	√D12	√	√				√				√					
2	√D13	√														√

After 6th Iteration R points to 5th node ,as 5th node is last item in for loop of iteration 4th R further moves one level up and points to node 3rd .Control returns to Step 7 of 2nd Iteration .

- Step 2 : $D_i = \{D3, D6, D13\}$
- Step 3 : Condition is false GOTO Step 4
- Step 4 : Condition is false GOTO Step 6
- Step 6 : Make D3,D6 and D13 children of D10. Mark tuples D3,D6 and D13
- Step 7 : for $t_j = D3, D6$ and $D13$. take $t_2 = D6$, $T = D - T = \{\phi\}$
- Step 8 : $R = t_2$, GOTO Step 2

In Step 7 D-T is empty as there is no element (tuple) in the table that has D6 as subset, similarly for D13. Control return to 1st iteration and $R = D10$.

Iteration 1st

- Step 1 : tag=1, R=NULL
- Step 2 : $D_i = \{D10\}$

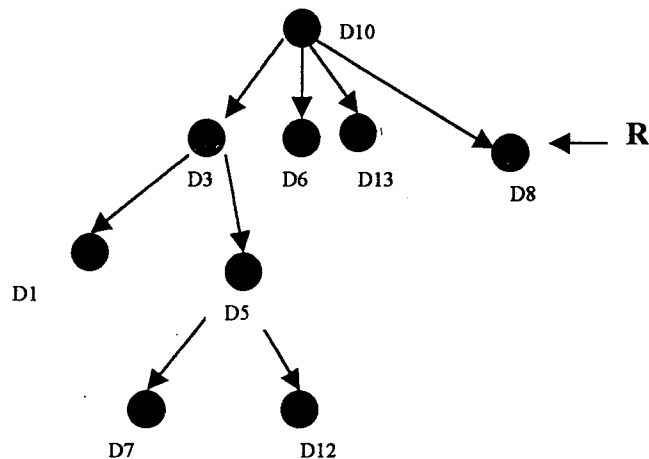
- Step 3 : Condition is false GOTO Step 4
 Step 4 : tag==1, GOTO Step 5
 Step 5 : Di contains one element so make it root (p=root), there is no non-participating element hence assign R=p; mark D10, tag=0, GOTO Step 7
 Step 7 : for tj=D10 do, t2=0
 T=D-T={D1,D2,D3,D4,D5,D6,D7,D8,D9,D11,D12,D13}
 Step 8 : j> number GOTO Step 9
 Step 9: Unmark tuple are present in the table GOTO Step 2

At this iteration R is pointing to D10 (root of HPRs tree)

7th Iteration

- Step 2 : Di={D8}
 Step 3 : Condition is false GOTO Step 4
 Step 4 : Condition is false GOTO Step 6
 Step 6 : Make D8 child of D10. Mark tuples D8
 Step 7 : for tj=D8. take t1=D8, T=D-T={ D2,D4,D9,D11}
 Step 8 : R=t1, GOTO Step 2

Following tree is obtained after 7th Iteration



The status of original Table is as follows where tuples D10,D3,D6,D13,D1,D5,D7 and D12 are marked.

Table 3.5

Count	Decision	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
3	√D1	√	√	√												
4	D2	√							√	√					√	
2	√D3	√	√													
4	D4	√				√			√	√						
3	√D5	√	√				√									
2	√D6	√			√											
4	√D7	√	√				√	√								
3	√D8	√							√	√						
6	D9	√							√	√			√	√	√	
1	√D10	√														
5	D11	√							√	√		√			√	
4	√D12	√	√				√				√					
2	√D13	√														√

8th Iteration

- Step 2 : $D_i = \{D2, D4\}$
- Step 3 : Condition is false GOTO Step 4
- Step 4 : Condition is false GOTO Step 6
- Step 6 : Make D2 and D4 children of D8. Mark tuples D2 and D4
- Step 7 : for $t_j = D2$ and $D4$ take $t_1 = D2$ $T = D - T = \{D11, D9\}$
- Step 8 : $R = t_1$, GOTO Step 2

9th Iteration

- Step 2 : $D_i = \{D11\}$
- Step 3 : Condition is false GOTO Step 4
- Step 4 : Condition is false GOTO Step 6
- Step 6 : Make D11 child of D2 Mark tuples D11
- Step 7 : for $t_j = D11$. take $t_1 = D11$ $T = D - T = \{\phi\}$
- Step 8 : $R = t_1$, GOTO Step 2

In Step 7 D-T is empty as there is no element (tuple) in the table that has D11 as subset.

j in t_j becomes 2.

10th Iteration

Step 2 : $D_i = \{\phi\}$
Step 3 : Condition is true GOTO Step 7
Step 7 : for $t_j = D_{11}$. Now j is 2
Step 8 : Condition is false i.e., GOTO Step 9 :
Step 9 : unmark tuple is present GOTO Step 2

In Step 9 Table consist of two tuples i.e., D_{11} and D_9 . Not the original Table

11th Iteration

Step 2 : $D_i = \{D_9\}$
Step 3 : Condition is false GOTO Step 4
Step 4 : Condition is false GOTO Step 6
Step 6 : Make D_9 child of D_2 . Mark tuples D_9
Step 7 : for $t_j = D_9$. take $t_1 = D_9$ $T = D - T = \{\phi\}$
Step 8 : $R = t_1$, GOTO Step 2

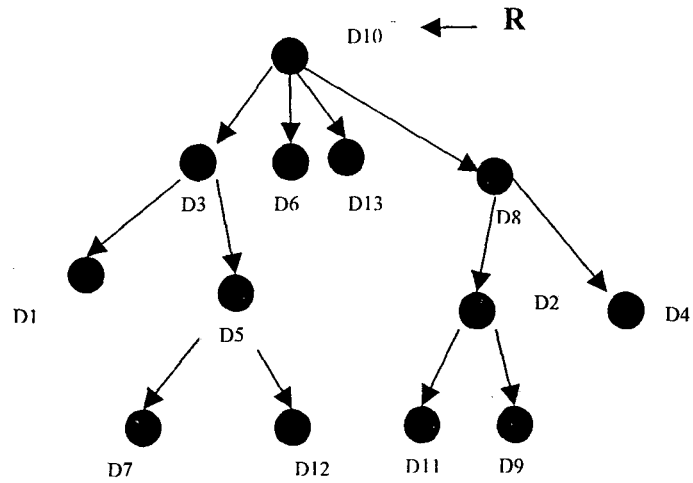
In Step 7 $D - T$ is empty as there is no element (tuple) in the table that has D_9 as subset. So, control returns to Step 7 of 8th Iteration

Step 2 : $D_i = \{D_2, D_4\}$
Step 3 : Condition is false GOTO Step 4
Step 4 : Condition is false GOTO Step 6
Step 6 : Make D_2 and D_4 children of D_8 . Mark tuples D_2 and D_4
Step 7 : for $t_j = D_2$ and D_4 take $t_2 = D_4$, $T = D - T = \{\phi\}$
Step 8 : $R = t_1$, GOTO Step 2

12th Iteration

Step 2 : $D_i = \{0\}$
Step 3 : condition is true GOTO Step 7
Step 7 : Now j is 2, $D - T = \{\phi\}$
Step 8 : condition is false i.e., GOTO Step 9 :
Step 9 : No unmark tuple is present GOTO Step 10
Step 10: Stop

Final tree obtained is shown below.



The status of original Table is as follows where all the tuples are marked

Table 3.6

Count	Decision	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15
3	√D1	√	√	√												
4	√D2	√							√	√					√	
2	√D3	√	√													
4	√D4	√				√			√	√						
3	√D5	√	√				√									
2	√D6	√			√											
4	√D7	√	√				√	√								
3	√D8	√							√	√						
6	√D9	√							√	√			√	√	√	
1	√D10	√														
5	√D11	√							√	√		√			√	
4	√D12	√	√				√				√					
2	√D13	√														√

Based on the tree generated. Following Hierarchical Production Rules can be formed

Hierarchical Production Rules

Rule 1 : D10 IF [P1]
GENERALITY [NULL]
SPECIFICITY [D3 D6 D13 D8]

Rule 2: D3 IF [P2]
GENERALITY [D10]
SPECIFICITY [D1 D5]

Rule 3 :D1 IF [P3]
GENERALITY [D3]
SPECIFICITY [NULL]

Rule 4:D5 IF [P6]
GENERALITY [D3]
SPECIFICITY [D7 D12]

Rule 5: D7 IF [P7]
GENERALITY [D5]
SPECIFICITY [NULL]

Rule 6:D12 IF [P10]
GENERALITY [D5]
SPECIFICITY [NULL]

Rule 7: D6 IF [P4]
GENERALITY [D10]
SPECIFICITY [NULL]

Rule 8: D13 IF [P15]
GENERALITY [D10]
SPECIFICITY [NULL]

Rule 9: D8 IF [P8 P9]
GENERALITY [D10]
SPECIFICITY [D2 D4]

Rule 10: D2 IF [P14]
GENERALITY [D8]
SPECIFICITY [D11 D9]

Rule 11: D11 IF [P11]
GENERALITY [D2]
SPECIFICITY [NULL]

Rule 12: D9 IF [P12 P13]
GENERALITY [D2]
SPECIFICITY [NULL]

Rule 13: D4 IF [P5]
GENERALITY [D8]
SPECIFICITY [NULL]

3.5 Algorithm (Automatic Generation Of Hierarchical Structure)

Algorithm 3.2

INPUT : A relational table with hierarchical information in the attributes.

OUTPUT: Hierarchical structure of attribute values

Step1 : Form all combination of attributes in the table.

Step2: Pick up any one new combination of attribute (A,B)

Step3: Check hypothesis (A low level , B high level i.e., $A < B$)

 For each value of B say B_i

 For every value of A say A_j

 If (B_i occurs with more than one A_j) then

 Hypothesis is wrong i.e., $B < A$

 If successful hypothesis is correct ($A < B$)

Step 4: Store the relation between attribute A and B. No combination left go to Step 5, else go to step2

Step 5: The pairs of attributes obtained from above Steps form the precedence graph

Step 6: Using topological sorting find order among attributes

Step 7: Scan the table and find the attribute value of attribute at level $l+1$ that Occurs together with attribute value of attribute at level l in the table, where $l=1, \dots, \text{max_att}$.

Consider the example 3.4 below[4]

Example 3.4

FOOD

Bar_code	Category	Brand	Content
452192	Milk	Foremost	3%
324091	Milk	Amul	3.5%
209649	Milk	Dairyland	3%
120097	Milk	Foremost	3%
213905	Milk	Paras	3.5%
908745	Milk	Dairyland	3%
249844	Milk	Paras	3.5%
356371	Bread	Wonder	0.32
120980	Bread	Safal	1.8%
567843	Bread	Britania	0.32%
289054	Bread	Motherdairy	1.8%
107922	Bread	Safal	1.8
128903	Bread	Wonder	0.32

Table 3.7

The algorithm will generate hierarchical structure of fig 3.4 for the given table

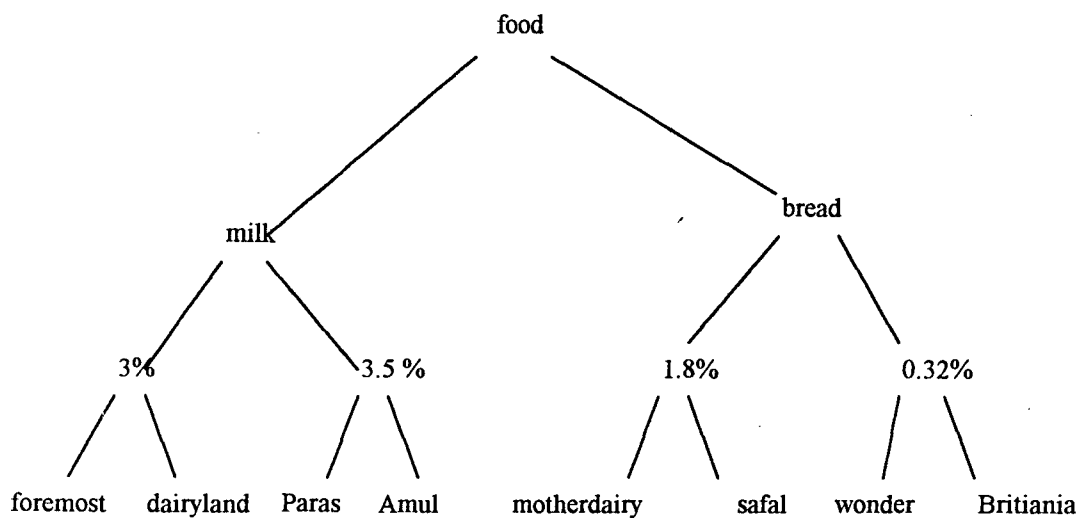


Fig 3.4 Hierarchical structure for table 3.7

Explanation of Algorithm

Step1 produces the following pairs of attributes { category,brand},{category , content},{brand ,content}

Choose {brand,content} and assuming brand < content,since 3% occurs with foremost and dairyland so our assumption is wrong and correct order is (content<brand.)

Similarly the correct order for other pairs are(category<brand)and(category<content).

Topological sort on these partial order will produce category<brand<content.

Table 1 implies concept tree of fig 1 where root is the name of the table.

Limitation of the algorithm

1. Hierarchy must exist in the table
2. A node must contain exactly one parent node.

The example 3.1 would not produce the correct hierarchy, if the same fat-content is present in more than one food category.

In practical above situation is common to occur in the food industry.

To extend the algorithm to cater the above mentioned restriction requires, a preprocess step that eliminates the noise and generate hierarchical structure with threshold support factor .

For example

Table 3.8 contains the extended table

1. In 4 tuples 3% and milk occurs together, and in a tuple 3% and bread appears together. So, we can take 3% and milk together with threshold support of $4/5$. eliminate tuple with 3% and bread.
2. Dairyland appears twice with 3% and once with 3.5%. So eliminate tuple with dairyland and 3.5%. Take Dairyland and 3% with threshold support of $2/3$.
3. Wonder appears once with 1.8% and twice with 0.32%. So, threshold Support factor of 0.32% with wonder is $2/3$.

Removing the tuples will give a relation that exhibits hierarchical structure with support factor of $2/3 = \min(4/5, 2/3, 2/3)$.

The obtained hierarchy is same as in fig 4.1 with support factor of $2/3$.

Bar_code	Category	Brand	Content
290876	Bread	Foremost	3
209872	Milk	Dairyland	3.5%
100922	Bread	Wonder	1.8%

Table 3.8 Extended table

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Introduction

In this chapter implementation and results of the algorithms for generation of HPR and automatic generation of hierarchical structure from large databases are presented. These two algorithms are discussed in the previous Chapter.

4.2 Generation of HPRs

Implementation

The table used in the algorithm is organised in the primary memory by a two dimensional array. Mark or unmark of a cell is represented as 1 or 0. The table is initialized to 0. The information about the premises of a rule is stored by assigning 1 to the corresponding cell in the table.

A structure with two fields count and mark is taken. Count field stores the number of 1's in a tuple in the table. Mark field is used to keep track of the tuples that has been processed. '1' and '0' in the mark field indicates processed and unprocessed respectively. As the tuples are processed a node for it is created in the primary memory, the general tree grows as the tuples are processed.

Finally using the tree formed in the memory, hierarchical production rules are generated.

Results

consider following set of production rules for different geometrical figure

Rules1: **If** 2-dim, closed_fig **Then** Plane_fig

Rule 2 : **If** 2-dim, closed_fig ,at_least_one_edge_is_not_a_straight_line **Then**
non_polygon

Rule 3 : **If** 2-dim, closed_fig , each_edge_is_a_straight_line **Then** polygon

Rule 4 : **If** 2-dim, closed_fig , each_edge_is_a_straight_line, 3-vertices
Then triangle

Rule 5 : **If** 2-dim, closed_fig , each_edge_is_a_straight_line, 4_vertices,all_edges
_equal,all_angles_equal **Then** square

When these 5 rules are provided as input to the algorithm 3.1, fig 4.1 is obtained. Further HPRs of fig 4.2 is obtained from fig 4.1.

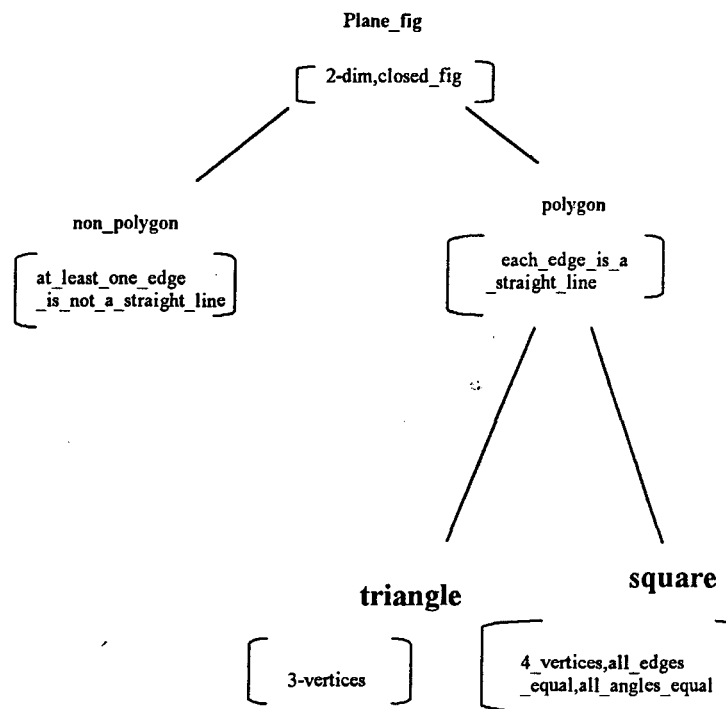


Fig 4.1 HPRs tree for plane figure

Plane_figure	IF[2-dim,closed_fig] GENERALITY[] SPECIFICITY[non_polygon,polygon]
Non_polygon	IF[at_least_one_edge_is_not_a_straight_line GENERALITY[plane_fig] SPECIFICITY[]
Polygon	IF[each_edge_is_a_straight_line] GENERALITY[plane_fig] SPECIFICITY[triangle,square]
Triangle	IF [3-vertices] GENERALITY[polygon] SPECIFICITY[]
Square	IF[4vertices,all_edges_equal,all_angles_equal] GENERALITY[polygon] SPECIFICITY[]

Fig 4.2 HPRs for fig 4.1

4.3 Automatic Generation Of Hierarchical Structure

Implementation

Large database is divided into number of small fixed size of data that can be accomodated in the primary mememory at a time. All possible combination of attribute are taken , by processing table within two for loops

```

for(att=1;att<max_att;att++)
  for(att_1 =1;att_1<=max_att;att_1++)
  {
    //body
  }

```

Partail order between pairs of attributes are generated using the propsed algorithm. opological sorting is performed on the set of partial order. This gives partial order among the attributes.

After obtaining partial order among attributes database is scanned once more to find the attribute value of attribute at level $l+1$ that Occurs together with attribute value of attribute at level l in the table, where $l=1, \dots, \text{max_att}$.

Topological sorting is implemented by an algorithm given below.[9]

The algorithm goes as follows.

The algorithm uses a sequential table $x[1], x[2], \dots, x[n]$, and each node $x[k]$ has the form

+	0	count[k]	top[k]
---	---	----------	--------

here $\text{count}[k]$ is the number of direct predecessors of object k (i.e., the number of pairs $j < k$ which have appeared in the input), and $\text{top}[k]$ is a link to the beginning of the list of direct successors of object k . The latter list contains entries in the format

+	0	suc	next
---	---	-----	------

where suc is a direct successor of k and next is the next item in the list. As an example of these convention, Fig 4.3 shows the schematic content of memory corresponding to the input

$$\begin{array}{l}
 9 < 2, \quad 3 < 7, \quad 7 < 5, \\
 5 < 8, \quad 8 < 6, \quad 4 < 6, \\
 1 < 3, \quad 7 < 4, \quad 9 < 5, \\
 2 < 8.
 \end{array}
 \tag{4.1}$$

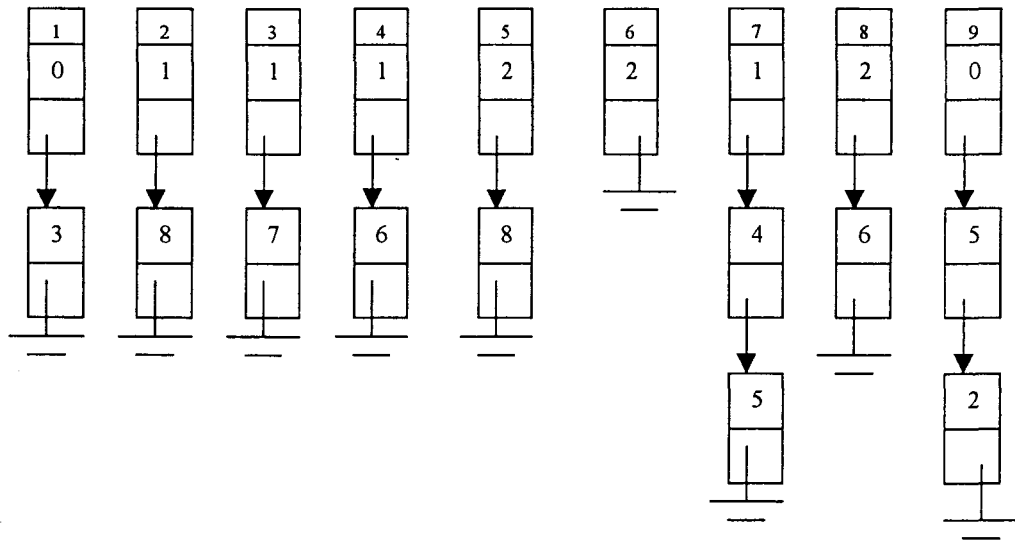


Fig 4.3 The schematic content of memory corresponding to the input 4.1

It is matter of outputting the nodes whose count field is zero , then decreasing the count fields of all successors of those nodes by one. The trick is to avoid doing any “searching” for nodes whose count field is zero, and this can be done by maintaining a queue containing those nodes whose count field has been reduced to zero but which have not yet been output. The links for this queue are kept in the count field , which by now has served its previous purpose.

Result

Here an example is provided.

consider the following data in fig 4.4. There is an inherent structure present in the data, When the data in fig 4.4 is provided as input to the algorithm 3.2 fig 4.5 is obtained.

- s.no**
1. A2 B4 C8
 2. A1 B2 C4
 3. A1 B1 C2
 4. A1 B1 C1
 5. A1 B2 C3
 6. A2 B4 C8
 7. A1 B2 C3
 8. A2 B3 C6
 9. A2 B3 C5
 10. A2 B4 C7
 11. A1 B1 C2
 12. A2 B3 C6
 13. A1 B1 C1
 14. A3 B5 C11
 15. A3 B5 C12
 16. A3 B5 C11
 17. A3 B6 C9
 18. A3 B6 C9
 19. A3 B6 C10

Fig 4.4

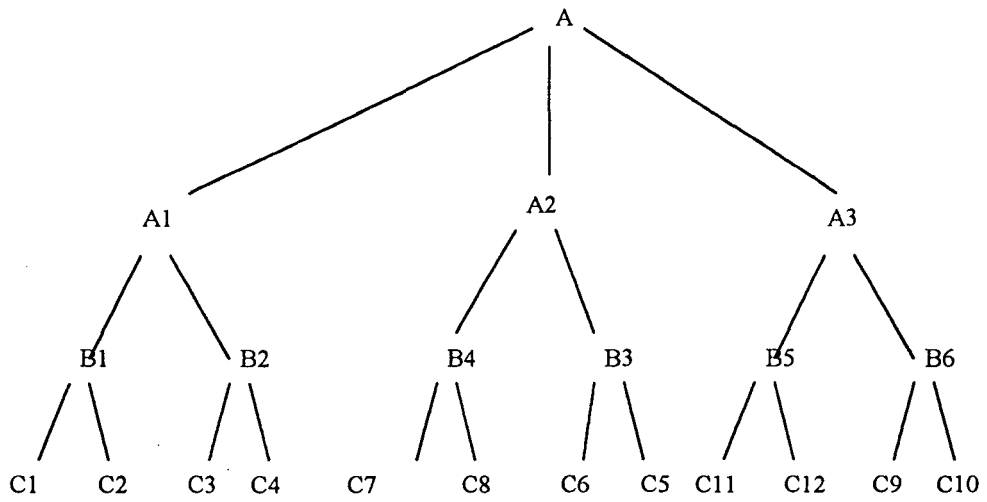


Fig 4.5 Hierarchical structure for data in fig 4.4

CHAPTER 5

CONCLUSION

As an attempt towards automatic generation of hierarchical structure an algorithm is developed that organizes the standard production rules into hierarchical structure that can further be used to generate Hierarchical Production Rules. Also an algorithm for automatic generation of hierarchical structures from large databases is proposed. Experimental results are presented to demonstrate working of the proposed algorithm.

An important extension of this work would be automatic generation of Hierarchical Censored Production Rules (HCPRs) from large databases.[7][8]

BIBLIOGRAPHY

- [1] J.Han and Y.Fu "Dyanmic generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases".In Proc AAAI' 94 workshop on knowledge discovery in databases(KDD' 94). Seattle . WA,157-168,1994
- [2] Text Book , Data Mining: Concepts and Techniques by , Jiawei han, Micheline Kamber © 2001 by Academic Press,ISBN 81-7867-023-2.
- [3] Agrawal, R; Imielinski, T.; and swami. A 1993."Mining Association rules between set of items in large databases" . In Proc of the ACM SIGMOD conference on management of dta,207-216.
- [4] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," Proceedings of the KDD, Newport Beach, CA, August 1997, pp. 67-73. 135
- [5] R.Agrawal and R. Srikant. "Fast algorithm for mining association rules". In proc. 1997 Int. conf.Very large databases,PP 487-499, stantiago,chile,sept 1994.
- [6] Han,J. And Fu Y 1995. "Discovery of multiple level association rules from large databases".In Proc.of the 21st Int'l conference on very large databases.
- [7] K. K. Bharadwaj and N. K. Jain "Hierarchical censored production rules(HCPRs) system". Data and Knowledge Engineering,8,19-34(1992)
- [8] N. K. Jain and K.K Bharadwaj "Some Learning Techniques in Hierarchical Censored Production Rules(HCPRs) System. Int'l Journal of intelligent systems, Vol. 13, 319-344(1998)
- [9] Art of Computer Programming, Volume 1: Fundamental Algorithms, 3/E, by Donald E. Knuth, © 1997 by Addison Wesley Professional, ISBN: 0-201-89683-4

