FINDING INTERESTING SUMMARIES IN GENSPACE GRAPHS

A Thesis

Submitted to the Faculty of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in Computer Science

University of Regina

By

Liqiang Geng

Regina Saskatchewan

August 2005

CHAPTER 1
INTRODUCTION

In this thesis, we address the problem of automatically finding interesting summaries of a data set. We propose an interactive summary mining approach, called ***GenSpace summary mining*** (***GSSM***), for finding interesting summaries based on belief revision. The remainder of this chapter is as follows. In Section 1.1, we define the summarization problem, discuss summarization in the context of Online Analytical Processing (OLAP) systems, and briefly identify the limitations of existing approaches to summary mining in OLAP systems. In Section 1.2, we introduce the concept of belief revision and propose a framework for using belief revision in data mining. In Section 1.3, we describe the GSSM problem in terms of its inputs, constraints, outputs, and challenges. In Section 1.4, we outline the remainder for the thesis.

1.1 Summarization in OLAP Systems

Summarization refers to the formation of interesting, compact descriptions of data. It was listed by Fayyad et al. as one of the six primary data mining tasks [Fayyad et al., 1996]. Finding interesting summaries for a large data set at different conceptual levels is a basic task for a decision-making process. For example, a manager of a chain of grocery stores might be presented with summaries of sales information for a month or a year, for a city or a country, for customers of different ages or with different incomes, and for categories of commodities. When the number of summaries is overwhelming, automatically finding the most interesting ones is desirable. Based on these interesting summaries, the CEO might gain insight into the performance of the company and then improve sales policies or assign resources to improve the performance of the company in the future.

Currently, the summarization problem is addressed by data warehouses and online analytical processing. A ***data warehouse*** is an analysis-oriented structure that stores a large collection of subject-oriented, integrated, time variant, and non-volatile data [Chaudhuri and Dayal, 1997]. ***Online analytical processing*** (***OLAP***) is a data analysis technique for processing multi-dimensional data warehouses. OLAP is widely used in applications such as marketing and sales analysis, financial reporting, and quality analysis [Chaudhuri and Dayal, 1997]. A common data model for OLAP is the ***data cube***, which is composed of a set of summaries at different conceptual levels. An OLAP system provides functionalities and operators to assist analysts in exploring the space of the data cube.

Although OLAP is a very powerful and practical technique, it suffers from some drawbacks. First, during exploration, the data analyst must examine summaries one by one to assess them, at least at the beginning of the exploration. However, the number of summaries that can be extracted from a data cube is exponential in the number of attributes. An attribute describes a feature of objects, associated with a set of values. In the summary tables, an attribute is represented by a column. As the number of attributes and the number of conceptual levels for each attribute increase, exploring the data cube with the basic operators becomes tedious and error-prone.

Secondly, OLAP cannot incorporate a user's estimated probability distributions for summaries, hence cannot compare them with the mined results and customize the presentation of results for users. For example, if a summary at a certain level matches a user’s estimated probability distribution, the system still presents it to the user. It is in the user’s discretion to decide which summaries are interesting to him/her.

Thirdly, the ability to respond to changes in the user's knowledge during the knowledge discovery process is limited. For example, if the information that more Pay-TV shows are watched during the evening than the afternoon has already been presented to the user, it will subsequently be less interesting to the user to learn that more Pay-TV shows are watched starting at 8:00PM than shows starting at 4:00PM. OLAP methods do not adjust their presentation based on changes in the user’s knowledge.

1.2 A Framework of the Integrated Data Mining and Belief Revision

The problem of modelling changes in a user’s knowledge can be regarded as a belief revision problem. Belief revision is an active research area in philosophy and artificial intelligence. Two of the most common frameworks for this problem are based on logical inference [Katsuno and Mendelzon, 1991; Val and Shoham, 1994; Herzig and Rifi, 1999] and probability inference [Jensen, 1996; Xiang, 2002]. Under both frameworks, the input to the problem is the initial belief *Bel* and new evidence ξ. In the logical inference framework, *Bel* and ξ are represented as logic formulas, while in the probability inference framework, they are represented as probability distributions. In both frameworks, the output is the revised knowledge base *Bel′*. The ***belief revision problem*** is to find an operator or reasoning method *O* to revise *Bel* according to the new evidence, such that $Bel' = O(Bel, \xi)$.

Belief revision and data mining can complement each other in the following way. First, when people are involved in the updating procedure and the amount of the evidence is overwhelming, using data mining techniques to select the most interesting evidence to revise the belief is attractive. On the other hand, incorporating the revised user's beliefs in the data mining process can improve the efficiency of the interaction during the mining process.

Figure 1.1 shows the framework of an integrated data mining and belief revision system. The initial belief set *Bel* is set by the user. The data mining technique acts on *Bel* and the evidence data ξ to select some interesting evidence ξ' and present it to the user, who adjusts his/her beliefs *B* accordingly and presents them to an updating system that revises the initial belief set. The process can then be repeated with other interesting evidence.

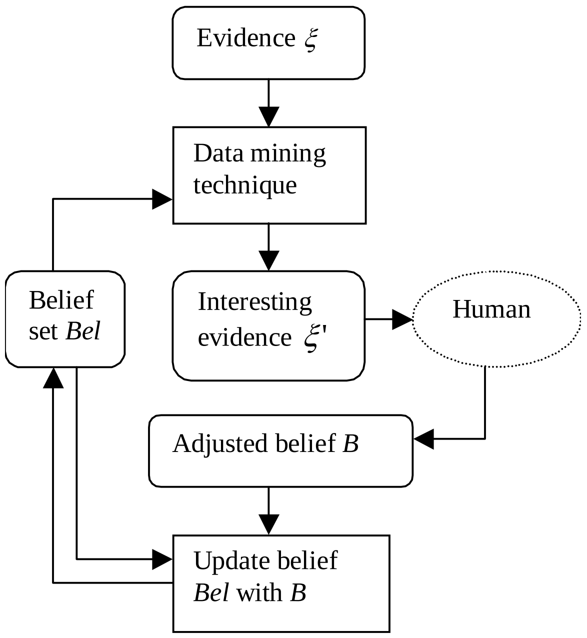


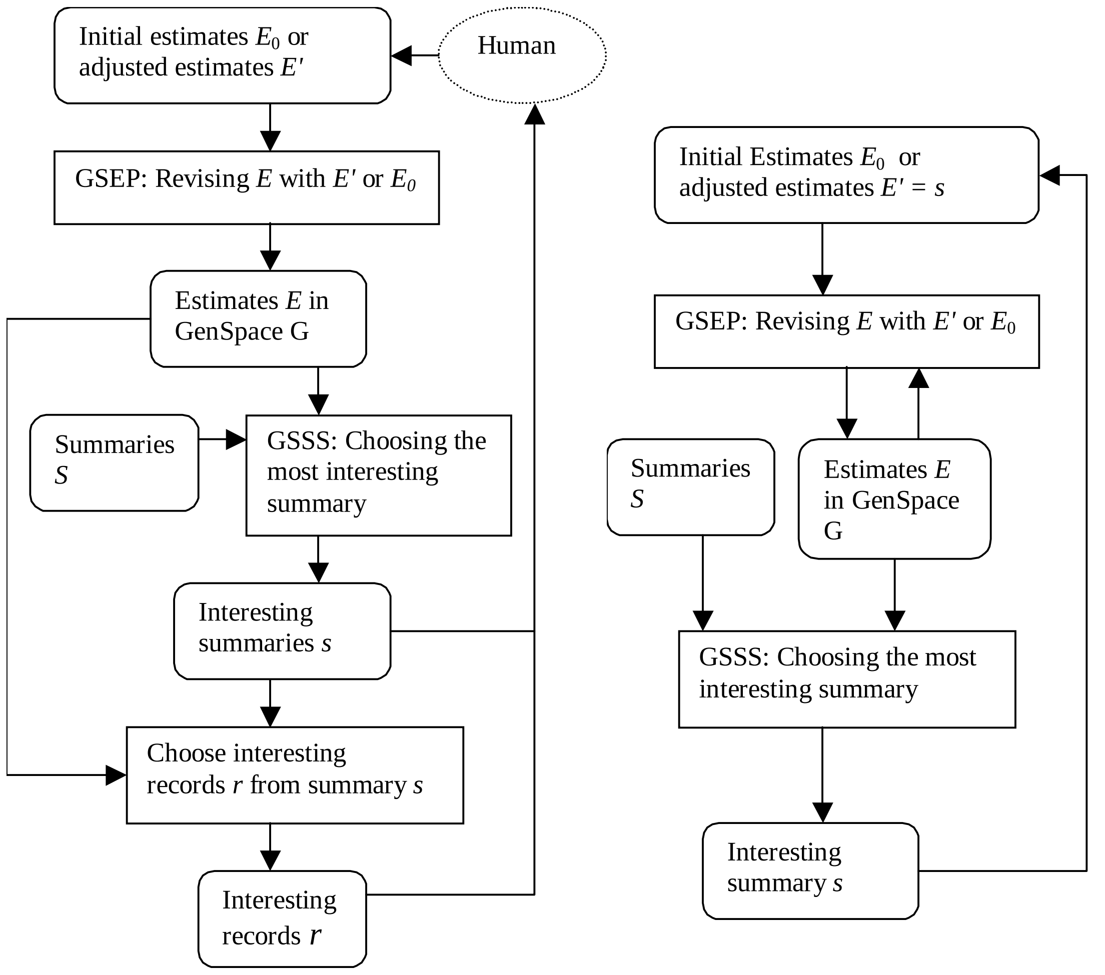
Figure 1.1 Framework for integrated data mining and belief revision

1.3 Objectives and Challenges

The summary mining problem addressed in this thesis, called the ***GenSpace summary mining*** (***GSSM***) problem, can be considered as an integrated data mining and belief revision problem, where the evidence is represented by the observed probability distribution, in the form of histogram data, of possible values at some conceptual levels (summaries) observed from a data set and the user's beliefs are represented by ***estimates***, i.e., estimated probability distributions, also in the form of histogram data, at a certain conceptual level in a graphical structure called a ***Generalization Space graph***, or ***GenSpace graph*** for short. A GenSpace is composed of a set of all possible summaries at different conceptual levels for multiple attributes, associated with the user's estimates. A GenSpace graph can be represented in compact form as a set of ExGen graphs, one for each attribute. An ExGen graph represents the generalization relations between conceptual levels for a single attribute.

The objective of this thesis is to provide a solution to the GSSM problem. The GSSM problem can be decomposed into the ***GenSpace summary selection*** (***GSSS***) problem and the ***GenSpace estimate propagation*** (***GSEP)*** problem. The GSSS problem is to find an interesting summary far from the user's estimates in a GenSpace graph, or in other words, to find summary s such that $s = SelectSummary(S, E)$, where E denotes the old estimates, S denotes the set of summaries, and *SelectSummary* chooses the most interesting summary s in S. The selected summary is used as the evidence for belief revision. The GSEP problem is to consistently revise the model of the user's estimates in the GenSpace graph by incorporating adjusted estimates caused by the evidence, using the formula $E' = O(E, \mathit{HumanChoice}(\mathit{SelectRecords}(s, E)))$, where E' denotes the new estimates in the GenSpace graph. The *SelectRecords* function chooses interesting records from an interesting summary. The *HumanChoice* function denotes the human choice of new estimates based on the interesting records presented to him/her. Both the *SelectRecords* and *HumanChoice* functions are optional in our study. If we do not consider them, we obtain the formula for the automated GSEP problem $E' = O(E, s)$.

Figure 1.2 shows both the human assisted and automated processes for the GSSM problem. In Figure 1.2(a), the user specifies the initial estimates E_0 for a given conceptual level in the ExGen graphs. The system propagates them to the entire GenSpace graph and obtains the estimates in all conceptual levels. Based on the estimates and the observed histogram data in summaries S, the most interesting summaries s are chosen and presented to the user. Furthermore, the most interesting records in the summaries can also be chosen to the user. The user then can adjust his estimates at a certain conceptual level and input it to the system. The system will start another round of propagation and mining. The process stops either when there are no more interesting summaries for presentation or when the user halts the process. Figure 1.2(b) is a simplified version of Figure 1.2(a), when we use the histogram data in the most interesting summary in the last round rather than the user’s input as the adjusted estimates in the next mining round. This method will be used in our experiments to demonstrate and evaluate the effectiveness of the GSSM method.


Initial estimates E_0 or adjusted estimates E'
Human
GSEP: Revising E with E' or E_0
Estimates E in GenSpace G
Summaries S
GSSS: Choosing the most interesting summary
Interesting summaries s
Choose interesting records r from summary s
Interesting records r
Initial Estimates E_0 or adjusted estimates $E' = s$
GSEP: Revising E with E' or E_0
Summaries S
Estimates E in GenSpace G
GSSS: Choosing the most interesting summary
Interesting summary s

(a) Human involved GSSM process (b) Automated GSSM process

Figure 1.2 Find interesting summaries to revise the user's estimates

The inputs of the GSSM problem are (1) a two-dimensional table to be summarized, (2) ExGen graphs for individual attributes in the table, which store the generalization/summarization relations for the attributes, (3) a user's estimates for a given node for each attribute, and (4) uninteresting nodes in ExGen and GenSpace graphs specified by the user. Two principles are used to generate constraints on the method: (1) the user's estimates among all levels of the summaries should be consistent and (2) the changes made to the estimates by the propagation method should be minimal. The outputs are the interesting summary records. A summary record is ***interesting*** if (1) the interestingness score of the summary is above a given threshold and (2) the chi-square test of the summary record is significant.

There are three challenges in dealing with the GSSM problem. First, we need an efficient and consistent method for dealing with the GSEP problem, i.e., propagating a user's adjusted estimates to various conceptual levels in a GenSpace graph. Second, we need to improve the efficiency of the GSSM process in terms of both storage space and time. Third, we need to seek appropriate subjective interesting measures for summaries since no research work has been done in this area up to date. To address the first problem, we propose the ***linear GenSpace estimate propagation*** method (***LGSEP***), which first propagates the user's estimates to the bottom node of the GenSpace graph and then propagates them upward to the entire graph. This method is simple, consistent, and preserves former knowledge to some extent. To address the second problem, we propose various pruning strategies for the different mining processes, including pruning uninteresting nodes in ExGen graphs, pruning uninteresting nodes in GenSpace graphs, and pruning nodes based on interestingness measures. To address the third problem, we study the objective interestingness measures that are suitable to incorporate estimates and convert them to subjective measures for GSSM. We also study their properties in the context of GSSM, especially those that can be used in pruning.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we review related work. We first compare the characteristics and functionality of data mining and OLAP. We then review and compare five existing aggregation methods for calculating summaries. Lastly, we review the interestingness measures, including objective measures, subjective measures, measures for association rules, and measures for summaries.

In Chapter 3, we describe the GSSM framework, formalize the GSEP problem, identify principles for GSEP, and propose a linear GSEP method that consistently and efficiently propagates a user’s estimates throughout a GenSpace graph.

In Chapter 4, we propose two methods to determine GSEP propagation paths for a GenSpace subgraph to reduce the storage and propagation costs. Determining the GSEP propagation path for a GenSpace subgraph is a variant of the constraint Steiner tree problem. After the user identifies uninteresting nodes, we can remove some of them. However, if we remove all of them, the propagation might be disrupted or the propagation costs might be increased. Of the two methods, the first method preserves the one step generalization property of the GenSpace. It uses a simpler data structure and algorithm for propagation, and it needs less time for generalizing one record in a summary to a higher conceptual level. Its disadvantages are that it usually keeps more nodes and has higher storage and scanning costs. The second method does not preserve the one step generalization property, and hence has the opposite advantages and disadvantages.

In Chapter 5, we propose a method to construct virtual bottom nodes for a GenSpace graph to reduce the storage and propagation costs during the mining process while obtaining the same results as with the original GenSpace graph. This method is based on the concept of granular computing. We present an algorithm to construct the virtual bottom node, analyze the factors that affect its efficiency, and prove invariance of the linear GSEP method when applied to a GenSpace graph with a real bottom node and the corresponding GenSpace graph with a virtual bottom node.

In Chapter 6, we describe the process of GSSS. We especially analyze the interestingness measures that are suitable for GSSS. We identified the monotonic property for Bray and Whittaker measure and the upper bound for Schutz, symmetric relative variance, and relative entropy measure. These properties can be used as pruning strategies for GSSS.

In Chapter 7, we present applications of our method. We apply the method to a Saskatchewan weather data set, a University of Regina student data set, and a customer data set. We also illustrate the mining process and present the results to show the effectiveness of our method.

In Chapter 8, we conclude the thesis and give suggestions for future research.

CHAPTER 2

BACKGROUND AND RELATED WORK

Two of the most challenging issues in solving the GenSpace summary mining problem are the efficiency of the summarization process and the selection of suitable interestingness measures. In this chapter, we review related work on these issues. First, in Section 2.1, we review Online Analytical Processing [Chaudhuri and Dayal, 1997], in which summarization is a key issue, and the work to extend the functionality of OLAP with data mining techniques. Then, in Section 2.2, we review some efficient algorithms for summarization. Finally, in Section 2.3, we review relevant interestingness measures in data mining.

2.1 Data Mining and Online Analytical Processing

Due to the widespread use of computers and the Internet, corporate, governmental, and scientific communities are overwhelmed with the influx of huge amounts of data. However, the raw data alone cannot benefit the decision making process for these organizations. Any value to be realized from this data will be obtained by analyzing those data and extracting useful patterns from them. ***Data mining*** and ***Online Analytical Processing*** (***OLAP***) are the two most prominent technologies for supporting strategic decision-making based on large data sets.

Data mining refers to the extraction of implicit, previously unknown, and potentially useful patterns [Fayyad et al., 1996]. Based on previous research work, we categorize the form of the patterns that can be mined into decision rules, association rules, instances (cases), clusters, summaries, anomalies, and graphic representations.

A ***decision rule*** is an "if-then" rule that describes a causal relationship between a set of condition attribute-value pairs and a decision class [Clark and Niblett, 1989; Grzymala-Busse, 1992; Quinlan, 1993]. The ***condition attributes*** describe the features of objects. The ***decision attributes*** describe the possible classes that the objects may belong to. This type of rule is useful for classification and prediction of future data. For example, a financial analyst might be interested in how a customer's income and assets would affect his/her credit rating. An ***association rule*** is defined as an if-then rule $X \rightarrow Y$, between two disjoint itemsets *X* and *Y*. An itemset is a set of items each of which can be considered as a binary attribute. An association rule is valid if it satisfies the *confidence* and *support* constraints [Agrawal et al., 1993; Brin et al., 1997; Bayardo, 1998; Han et al., 2004]. ***Support*** is defined as the fraction of the records in the data set containing the itemset *XY*. ***Confidence*** is defined as the fraction of the records containing the itemset *XY* to those containing the itemset *X*. Although association rules can also be represented in if-then form, they are different from decision rules. First, association rules are usually used as descriptive rather than predictive tools. They represent associations between two itemsets rather than a logic implication. Second, association rules do not distinguish decision attributes from condition attributes. Third, the items are always represented by binary attributes. A typical use of association rules is to find sales relations between any sets of commodities in a transaction database to form the sales and advertisement strategies. ***Instance based learning*** tries to find representative cases in dataset for classifying new instances [Aha et al., 1991; Smyth and McKenna, 1999; Wilson and Martines, 2000; Geng and Hamilton, 2002; Geng and Hamilton, 2003a]. The difference between instance based learning and rule based learning is that instance based learning uses similarity measures rather than exact matching for both mining and classification procedures. Thus it can be used for mining complex data types, such as images and unstructured and semi-structured texts**.** ***Clustering*** is a data mining method that makes cluster of instances that are similar in characteristics [Guha et al., 1998; Zhang et al., 1996; Kaufman and Rousseeuw, 1990; Agrawal et al., 1998]. An instance in a cluster should bear a stronger resemblance to the other instances in the same cluster than to those in other clusters. Unlike decision rule mining, clustering is unsupervised learning where no decision attributes are available. ***Summarization*** is the formation of interesting, compact descriptions of data at different conceptual levels [Beyer and Ramakrishnan 1999; Harinarayan et al., 1996; Sarawagi et al., 1996]. For example, a sales manager may be interested in the sales information of a certain region during a certain period of time at different granularity levels, such as based on month and province, or week and city. ***Anomalies*** are the most significant changes in the data from previously measured or normative values. [Bay and Pazzani, 2001; Webb et al., 2003]. For example, in the 1990s the student acceptance rate of a university was 20%, but in 1994, the acceptance rate was 40%, which is significantly different from that of the other years. In this case, the year 1994 is an anomaly and may be worthy of special concern. Graphic representations of the mined knowledge include decision trees [Quinlan, 1993], neural nets [Bishop, 1995], and Bayesian belief networks [Jensen, 1996]. A variety of algorithms have been designed to learn these patterns from data sets.

OLAP is a powerful data analysis method and technology for multi-dimensional data warehouses and widely used in various applications [Chaudhuri and Dayal, 1997].

The most popular data model for OLAP is multi-dimensional ***data cube***. A data cube is composed of a set of summaries, called ***cuboids***, which form a concept lattice. OLAP systems provide basic functionalities and operators to assist analysts in exploring data cubes. The ***rollup*** operator moves from more general summaries to more detailed ones. The ***drilldown*** operator moves from detailed summaries to more general ones. The ***slice*** operator performs a selection on one dimension of a data cube. The ***dice*** operator performs a selection on two or more dimensions of a data cube. The slice and dice operators help user select different facets of the data cube that are interesting to him/her for presentation.

Data mining and OLAP have much in common. First, both data mining and OLAP help an analyst find useful knowledge and support decision-making. Secondly, they both deal with large amounts of data, and hence computational efficiency is an important issue for them.

However, they have differences regarding their functionalities. First, although data mining methods can deal with concept hierarchies, the core techniques of data mining are still based on the raw data, while OLAP essentially deals with data with concept hierarchies. Secondly, data mining is more autonomous than OLAP. A data mining system usually formulates new hypotheses and chooses a relatively small set of interesting patterns to present to users according to user specified conditions. Conventional OLAP systems cannot suggest new hypotheses. The user must formulate a hypothesis and manually explore the data cube to verify his/her hypothesis. The user must also choose the path in the lattice for exploration. Since the number of cuboids is exponential in the number of the attributes, exploring the data cube with the basic operators can become too time consuming and error prone, as the number of the attributes and the number of concept levels for each attribute increase.

Due to these features of OLAP and data mining, much work has been done to integrate them to improve their functionality for decision support.

Han et al. proposed a framework to combine data mining functions and OLAP [Han, 1998]. They added an ***online analytical mining*** (***OLAM***) engine on top of the data cube, so that data mining tasks, such as association analysis, classification, prediction, and clustering analysis can be performed at different levels of the data cube. This type of integration is straightforward.

To reduce the storage and computation costs, the iceberg cube method has been proposed. An ***iceberg cube*** is a subset of a data cube containing aggregates whose aggregate values are greater than or equal to a given support threshold [Beyer and Ramakrishnan, 1999]. The iceberg cube method incorporates pruning technique used in data mining to improve the efficiency of OLAP systems. However, this approach assumes that only the summaries with high aggregate values are interesting, which is not always the case. For example, the sales of a company in Ontario are perhaps always greater than those in Saskatchewan, but that does not mean that the sales in Ontario are always more interesting to analysts than those in Saskatchewan if Saskatchewan's sales have dramatically increased while Ontario's have remained steady.

Discovery-driven exploration has been proposed to guide the exploration process by providing analysts with interestingness measures based on statistical models [Sarawagi et al., 1998]. Initially, the user specifies a starting data cuboid and a starting cell, and the tool automatically calculates three values for each cell based on statistical models. The first value indicates the interestingness of this cell itself, the second value indicates the interestingness it would have if the data were drilled down from this cell to more detailed cuboids, and the third value indicates which paths to drill down from this cell. The user can follow the guidance of the tool to navigate through the space of the data cube.

Sarawagi simplifies the discovery driven exploration process by automatically finding the underlying reasons for exceptions [Sarawagi 1999]. The user identifies an interesting difference between two cells. The system then presents the most relevant data in more detailed levels that account for the difference.

Fabris and Freitas defined interestingness measures for attribute-value pairs in a data cube [Fabris and Freitas, 2001]. For a single attribute, the measures reflect the difference from the uniform distribution. For the interaction of two attributes, the measures are based on the assumption that dependencies are of interest, i.e., the more they are correlated, the greater the value of the interestingness measure. Coefficients are used to add bias towards the higher-level concepts in the data cube. Based on the interestingness measures, a list of attribute-value pairs is outputted. Users can pay particular attention to these attribute value pairs when they explore the data cube.

In general, three approaches have been proposed to integrate the OLAP and the data mining techniques. First, OLAP can be taken as a platform to perform different data mining tasks. Secondly, the data mining technique can be used to improve the efficiency of OLAP process. Thirdly, the data mining technique can be used to guide and facilitate users’ exploration of the data cube.

2.2 Aggregation Methods in OLAP

The methods to calculate summaries in OLAP systems are called ***aggregation methods***. Aggregation methods have been intensively studied since the introduction of data warehousing, data cubes and OLAP. To improve the efficiency of aggregation methods, researchers have proposed techniques in the following categories.

(1) Search orders, including specific-to-general, general-to-specific, breadth first, and depth first [Agarwal et al., 1996; Ross and Srivastava, 1997].

(2) Heuristics, including Apriori pruning and the selection of the smallest parent for aggregation [Beyer and Ramakrishnan 1999; Harinarayan et al., 1996].

(3) Data structures to facilitate calculation, including arrays, hash tables, and trees [Agarwal et al., 1996; Ross and Srivastava, 1997; Han et al., 2001].

(4) Storage reduction techniques, including iceberg cubes and selection of views to materialize [Beyer and Ramakrishnan 1999; Harinarayan et al., 1996; Han et al., 2001].

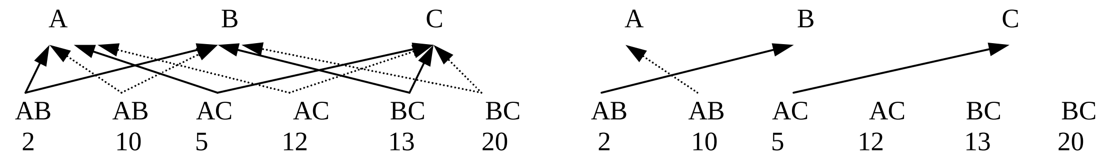
(5) Data rearrangement, include sorting and partitioning [Agarwal et al., 1996; Ross and Srivastava, 1997; Beyer and Ramakrishnan 1999].

We review several well-known algorithms that use techniques from these five categories. PipeSort is discussed in Section 2.2.1, Partitioned cube is discussed in Section 2.2.2, BUC is discussed in Section 2.2.3, Selecting views to materialize is discussed in Section 2.2.4, and Top-k cubing is discussed in Section 2.2.5.

2.2.1 PipeSort

Agarwal et al. proposed an efficient specific-to-general aggregation method called PipeSort [Agarwal et al., 1996]. It takes into account both share-sort and smallest-parent optimizations. ***Share-sort optimization*** aims at sharing the sorting results across multiple cuboids to improve aggregation calculation. ***Smallest parent optimization*** aims at computing cuboids from the smallest previously computed cuboids. The PipeSort algorithm first assigns two costs to each node in the search lattice. The first cost, called *S*, is the cost of computing a child (a node at a more general level) from this node when the node is not sorted. The second cost, called *A*, is the cost of computing a child from it when it is sorted according to the order of the child. For example, if node *AC* is calculated from node *ABC* and *ABC* is not sorted, the aggregation will have an *S* cost. If *ABC* is sorted in the order *ACB*, with *AC* as its prefix, the aggregation will have an *A* cost. From the most general node, in a level-by-level manner, the PipeSort algorithm finds the optimal parents for the child nodes using the weighted bipartite matching algorithm [Papadimitriou and Steiglitz, 1982]. In each level, if a child node is connected with a parent node with a cost *A*, the parent node will be sorted in the order of its child node. In this manner, a tree will be generated with two kinds of edges, pipeline edges and sort edges. An edge is a ***sort edge*** if before calculating the child, the parent needs to be sorted. Therefore, the cost of a sort edge is *S*. An edge is a ***pipeline edge*** if the parent is sorted in the order of its child, in other words, the child node is denoted as a prefix of its parent. Therefore, the cost of a pipeline edge is *A*. For each sort edge, the algorithm finds a path, called ***pipeline***, for aggregation.

Figure 2.1 shows an example of determining the path for calculating nodes *A*, *B*, and *C* from nodes *AB*, *AC*, and *BC*. The nodes *AB*, *AC*, and *BC* are assigned *A* costs of 2, 5, and 13, respectively. The algorithm then makes one copy of *AB*, *AC*, and *BC* and assigns them the *S* costs 10, 12, and 20, respectively, and connects them with *S* edges to the same set of child nodes. The number of the copies that should be made equals to the number of the attributes in the parent node minus 1. In Figure 2.1(a), the solid lines represent *A* edges and the dotted lines represent *S* edges. Now we need to select a subset of the parent nodes with the minimum cost such that it covers all the child nodes. After applying the weighted bipartite matching algorithm, we obtain the minimum cost matching as shown in Figure 2.1(b) (The solution to a bipartite matching may not be unique. In Figure 2.1, we only show one solution). Thus, the node *AB* will be sorted in the order *BA*, so that *B* is calculated from it without resorting. Node *A* is generated with resorting node *AB* in the order of *AB*. Node *AC* will be sorted in the order *CA* so that the node *C* will be calculated without resorting. Node *BC* is not used to calculate any child nodes; therefore, it can be sorted in any order.



(a) Bipartite graph (b) Aggregation path

Figure 2.1 Selecting an aggregation path with PipeSort

After processing all the levels, a tree is obtained for aggregation.

This method has some limitations. First, the authors did not explain how to estimate the *A* and *S* costs. Secondly, the method finds the local optimal parent for each node, but it may not be globally optimal. Thirdly, as pointed out by [Ross and Srivastava, 1997], the number of sorting procedures is not optimal. Fourth, the sorting method does not address the situation where concept hierarchies are involved.

2.2.2 Partitioned-Cube

To reduce the sorting cost and caching cost of Pipesort, Ross and Srivastava, 1997 proposed another specific-to-general algorithm to calculate data cubes [Ross and Srivastava, 1997]. The algorithm consists of two parts, Partitioned-Cube and Memory-Cube. Partitioned-Cube partitions the original relation into smaller groups according to attribute values to reduce the use of memory. It first checks if the original relation fits in memory. If it does, it calls Memory-Cube to aggregate and output the results. Otherwise, it partitions the relation into a set of smaller relations according to an attribute *A*. Then it uses the smaller relations as parameters to recursively call itself for each partition to get an aggregate for each value for attribute *A*. Then it recursively calls itself by projecting the attribute *A* out to get aggregates without attribute *A*. Finally, it unions all the results from the recursive calls.

Memory-Cube is similar to PipeSort method, except that it only focuses on the sorting cost (PipeSort also takes into account the size of the parent node). It is proved that for a lattice of *k* attributes, the minimum number of aggregation paths is $C_k^{[k/2]}$, each of which requires one sorting of the data. Memory-Cube finds the minimum number of aggregation paths, and therefore requires the minimum number of data sorting.

The advantage of Partitioned-Cube and Memory-Cube is that they use a divide and conquer method to compute data cubes without caching any of the results for later reuse. Thus they do not incur any additional I/O costs beyond the input of the relation and the output of the data cube itself. The limitations of the method are that it does not have pruning heurists and all the aggregates will be calculated and it does not address the situation where concept hierarchies are involved.

2.2.3 BUC

An iceberg cube is proposed to reduce the calculation cost and storage cost for a data cube because it only outputs the aggregates whose aggregate values are higher than a given threshold. Beyer and Ramakrishnan proposed a general-to-specific aggregation method, called ***bottom up computation*** (***BUC***), for sparse and iceberg cube [Beyer and Ramakrishnan 1999]. As with Apriori algorithm, pruning is based on the monotonic property of the measure: if the measure of a more general aggregate is below a given threshold, the measures for the corresponding more specific aggregates will be below the threshold, and thus can be pruned. The difference between BUC and Apriroi is that BUC uses a depth first strategy to traverse the space, while Apriori uses a breadth first strategy.

The traverse path and order for a four-attribute lattice is shown in Figure 2.2. The numbers in Figure 2.2 indicate the order in which BUC visits the cuboids. The algorithm first calculates the aggregate node *All*, if the measure is above the threshold, they it will calculate all the aggregates containing a value of attribute *A* in nodes 2-9. We assume there are two values for each attribute, a_1 and a_2 for *A*, b_1 and b_2 for *B*, and so on. The algorithm will first calculate the aggregate for a_1, then a_1b_1, $a_1b_1c_1$ and $a_1b_1c_1d_1$ in a depth first traversal. Then the algorithm will calculate the aggregate for $a_1b_1c_1d_2$. Then it backtracks to node *ABC* to calculate aggregate $a_1b_1c_2$. This process continues until all the aggregates containing a_1 is calculated. Then it calculates all the aggregates containing a_2 in the same manner. After all aggregates containing a value of *A* are calculated, it starts to calculate all aggregates containing *B* but not *A* in nodes 10-13, and then to calculate all aggregates containing *C* but not *A* and *B* in nodes 14-15. Finally it calculates all aggregates containing a value of attribute *D* but not *A*, *B*, and *C* in node 16. In a stage, if the measure of any aggregate is below a threshold, the algorithm will not move on to the more specific levels in the search tree and continues for the next aggregates in the current node.

The advantage of the depth first traversal used in BUC is that it needs less memory. The disadvantage is that the pruning may be one step latter than with Apriori. In the depth first search, when a_1b_1 is aggregated and its measure is above the threshold, $a_1b_1c_1$ needs to be aggregated next to see if it can be pruned. However, in the breadth first search, a_1c_1 is aggregated before $a_1b_1c_1$. If a_1c_1 is below the threshold, $a_1b_1c_1$ does not need to be aggregated.

The advantage of BUC (general-to-specific aggregation) over specific-to-general aggregation methods is that BUC can use pruning heuristics based on the iceberg cube threshold. A dramatic drawback of BUC is that it does not take advantage of aggregates that have already been obtained to reduce the aggregation cost. In BUC, every aggregate is aggregated from the most specific records.

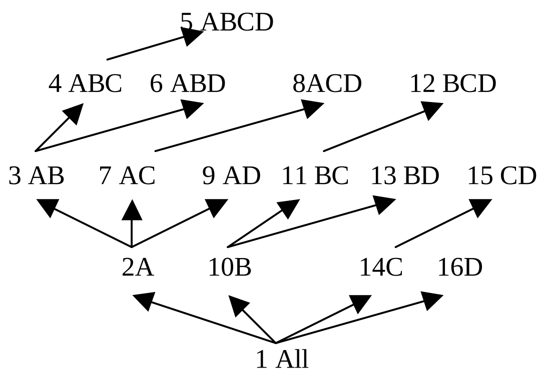


Figure 2.2 Aggregation path for BUC

2.2.4 Selecting Views to Materialize

When answering a query about cuboids, if the system does not materialize any cuboids, the average response time is high; if the system tries to materialize all cuboids, available memory might not be sufficient. Harinarayan et al. suggest a tradeoff between the time cost and the space cost [Harinarayan et al., 1996]. The suggested *greedy node selection algorithm* selects a subset of *k* cuboids to materialize while minimizing the average query time, where *k* is given by users. The authors first proposed a linear time cost model, which states that the cost of answering a cuboid *A* is directly proportional to the number of records present in the cuboid *B* that is used to construct *A*. With this model, they use the number of the records in *B* to represent the time cost to construct any decedents that are constructed from *B*.

In the greedy method to select nodes to materialize, they define the benefit $B(v, S)$ of a node *v* on a set *S* of selected nodes as follows.

1. For each descendent *w* of *v*, define $B_w(v, S)$ by

(a) Let *u* be *w*'s ancestor with smallest size in *S*.

(b) If cost(v) < cost(u), then $B_w(v, S) = cost(v) – cost(u)$, otherwise, $B_w(v, S) = 0$.

2. $B(v,S) = \sum_{w} Bw(v,S)$, where *w* is a descendent of *v*.

For each node *v* in the search lattice, the algorithm calculates the total benefit $B(v,S)$. Then it selects the node with the maximum benefit into the set *S*. This process is repeated until *k* nodes have been selected.

Figure 2.3 shows a search space for eight nodes. The numbers in the parentheses denote the sizes of the nodes, which is also the cost of constructing a descent from this node. If the user wants to select two nodes to materialize (besides base node a), the algorithm first calculates the potential benefit of nodes *b* to *h*. If *b* is selected, five nodes *b*, *d*, *e*, *g*, and *h* will benefit and each of these nodes will benefit by 50 records. Therefore, the total benefit of materializing *b* is *benefit*(*b*) = (100 – 50) * 5 = 250. Similarly *benefit*(*c*) = 125, *benefit*(*d*) = 160, *benefit*(*e*) = 210, *benefit*(*f*) = 120, *benefit*(*g*) = 99, *benefit*(*h*) = 90. The algorithm selects node *b* to materialize. With nodes *a* and *b* materialized, the algorithm calculates the benefit of materializing nodes *c* to *h* again. If we select node *f*, node *h* will benefit by $size(b) - size(f) = 10$ and node *f* will benefit by $size(B) - size(f) = 60$. The total benefit that node *f* yields is *benefit*(*f*) = 70. Similarly, we have *benefit*(*c*) = 50, *benefit*(*d*) = 60, *benefit*(*e*) = 60, *benefit*(*g*) = 49, and *benefit*(*h*) = 40. Since node *f* yields the maximum benefit, it is selected.

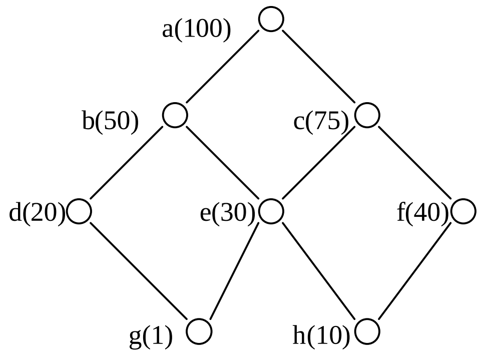


Figure 2.3 Selecting nodes to materialize

The authors also proved that the benefit achieved by the greedy algorithm is at least 1-1/*e* (63%) time of the benefit achieved by the optimal solution. This method can be applied to concept hierarchies.

2.2.5 Top-*k* Cubing Using a Hyper Tree Structure

Han et al. proposed an efficient algorithm to calculate iceberg cubes with measure average [Han et al., 2001]. Since average is not a monotonic measure, Apriori-like pruning cannot be applied during the general-to-specific calculation in the mining process. To exploit the Apriori property, the authors developed a mapping that transforms *average* to a weaker, but monotonic measure *top-k average* for pruning. A *top-k average* is the average value of the top-*k* base (most specific) cells of a cell when sorted in value-descending order. With this top-*k* average measure, pruning can be applied in the general-to-specific calculation. The authors also proposed a ***binning technique*** to reduce the storage and computation cost caused by the large value of *k* during the calculation of the top-*k* average.

Another contribution of this paper is that it uses a data structure H-tree to facilitate the computation. An ***H-tree*** is a compact representation of the data set that is built as follows. First, a scan of the original table creates a *header table,* in which each entry records the ***quant-info*** (including the *sum*, *count* and top *k* bins, that are needed to calculate the *average* and *top-k average*) for each attribute-value pair. Next, the root, labeled with *null*, of the tree is created. Then, it scans the table for the second time. For each record, it inserts the record into the tree. If the record matches a part of the path, it shares the common part of the path. If the record matches a whole path, then the *quant-info* in the leaf node will be updated. If two leaf nodes have the same label, it will be connected by a *side-link*. Figure 2.4 is a sample H-tree with its head table for Table 2.1 (Adapted from [Han et al., 2001]). The aggregation of the data cube is conducted by traversing and updating the H-tree structure.

Table 2.1 An example table

Month	Customer	City	Price
Jan	Edu	Toronto	500
Jan	House	Toronto	1000
Jan	Edu	Torontol	800
Feb	Busi	Montreal	700
Mar	Edu	Vancouver	200

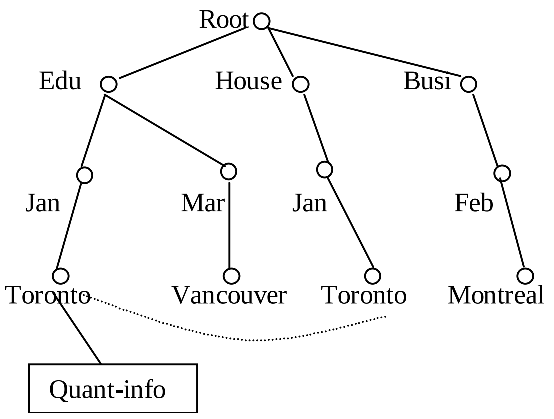


Figure 2.4 The H-tree structure for Table 2.1

From the example, we can see that each level of the tree corresponds to a variable. The H-tree is a compact representation of the original table and it preserves all information that is needed to calculate the average and top-k average measures. During the traversal, more specific cells of a cell are pruned if the top-k average measure of this cell is below the given threshold. The limitation of this method is that the tree cannot fit in memory when the search space of the H-tree is huge.

2.2.6 Comparison of the Aggregation Methods

Table 2.2 presents a comparison of five aggregation methods in summary form. The PipeSort and Partitioned-Cube are specific-to-general methods. They calculate aggregates from the most specific level to the most general level, and output the full cube. The advantage of these methods is that they can use sorting and partitioning and the calculated optimal intermediate cuboids to calculate the new cuboids. The disadvantage is that they do not use heuristics to prune the search space and cannot apply to the situations that concept hierarchies are involved. BUC and H-Tree methods are general-to-specific methods that are suitable to calculate iceberg cubes. They only output cells that meet some conditions. The advantage of these methods is that they use pruning methods by exploiting the monotonic property of the measures when calculating iceberg cubes. The disadvantage is that they need to calculate each summary from the original table and cannot use intermediate cuboids. They also do not take into account concept hierarchies. Selected materialization can be used where concept hierarchies are involved, but it does not materialize all the aggregates and only calculates the specified cuboids for a query.

Table 2.2 Comparison of aggregation methods

Algorithms	Search Order	Prunning Heuristics	Output	Applicable to Concept Hierarchies
PipeSort	Specific-General	No	All	No
Partitioned-Cube	Specific-General	No	All	No
BUC	General-Specific	Yes	Partial	No
Selected Materialization	Specific-General	No	Partial	Yes
H-Tree	General-Specific	Yes	Partial	No

2.3 Interestingness Measures

Measuring the interestingness of discovered patterns is an active and important area of data mining research. Although much work has been conducted in this area, so far there is no widespread agreement on a formal definition of interestingness in this context.

In this section, we review the interestingness measures used in different data mining tasks. We first describe the characteristics of interestingness patterns in Section 2.3.1. We then review objective interestingness measures in Section 2.3.2, and review subjective interestingness measures in Section 2.3.3.

2.3.1 Characteristics of Interesting Patterns

Based on the diversity of definitions presented to date, interestingness is perhaps best treated as a very broad concept, which emphasizes conciseness, coverage, reliability, novelty, surprisingness, peculiarity, utility, and actionability. These eight more specific criteria are used to determine whether or not a pattern is interesting as follows.

(1) Conciseness

A pattern is *concise* if it contains relatively few attribute-value pairs. A pattern set is concise if it contains fewer patterns. First, a concise pattern or pattern set is easier to understand and remember and thus is added more easily to the user's knowledge. Secondly, a more concise pattern tends to be more general and less sensitive to noise. In this sense, a more concise pattern is more interesting. Accordingly, much research has been done to find the "minimum set of patterns" using the properties of monotonicity [Padmanabhan and Tuzhilin, 2000] and confidence invariance [Bastide et al., 2000].

(2) Generality/Coverage

A pattern is *general* if it covers a relatively large subset of a dataset. *Generality* (or *coverage*) measures how comprehensive a pattern is, i.e., in what fraction of all cases the pattern occurs. If a pattern characterizes more information in the data set, it tends to be more interesting [Agrawal and Srikant, 1994; Webb and Brain, 2002]. Frequent item sets are the most studied general patterns in the data mining literature. An itemset is *frequent* if its *support*, the fraction of the records in the data set containing the itemset, is above a given threshold [Agrawal and Srikant, 1994]. The best known algorithm for finding frequent itemsets is the Apriori algorithm [Agrawal and Srikant, 1994]. Some generality measures can form the bases for pruning strategies; for example, the support measure is used in the Apriori algorithm as the basis for the pruning of itemsets. For classification rules, Webb and Brain gave an empirical evaluation showing how generality affects classification results. Generality frequently coincides with conciseness, because concise patterns tend to have greater coverage.

(3) Reliability

A pattern is *reliable* if the relationship described by the pattern occurs in a high percentage of applicable cases. For example, a classification rule is reliable if its predictions are highly accurate, and an association rule is reliable if it has high confidence. Many measures from probability, statistics, and information retrieval, have been proposed to measure the reliability of association rules [Ohsaki et al., 2004; Tan et al., 2002].

(4) Novelty

A pattern is *novel* to a person if he or she did not know it before and is not able to infer it from other known patterns [Dong and Li, 1998; Li and Hamilton, 2004]. Although some data mining systems represent what the user knows, no known data mining system represents what the user does not know. Thus, novelty cannot be measured explicitly with reference to the user's knowledge. Instead novelty is detected by either having the user explicitly identify a pattern as novel [Sahar, 1999] or by noticing that a pattern cannot be deduced from and does not contradict previously discovered patterns.

(5) Surprisingness

A pattern is *surprising* (or *unexpected*) if it contradicts a person's existing knowledge or expectations [Liu et al., 1997; Liu et al., 1999; Silberschatz and Tuzhilin, 1995; Silberschatz and Tuzhilin, 1996]. A pattern that is an exception to a more general pattern that has already been discovered can also be considered as surprising [Bay and Pazzani, 1999; Carvalho and Freitas, 2000]. Surprising patterns are interesting because they identify failings in previous knowledge, show new things, properties, or relationships, and may suggest an aspect of the data that needs further study.

(6) Peculiarity

A pattern is *peculiar* if it is far away from the other discovered patterns according to some distance measure. Peculiar patterns, also called *outliers*, are generated from peculiar data, which are relatively few in number and are significantly different from the rest of the data [Knorr et al., 2000; Zhong et al., 2003]. Peculiar patterns may be unknown to the user, and hence interesting. Peculiar pattern discovery is useful in fraud detection.

(7) Utility

A pattern is of *utility* to a person if its use by that person contributes to reaching a goal. Different people may have different goals concerning knowledge that can be extracted from a data set. For example, one person may be interested in finding the sales with the most profit in a transaction data set. Another person may be interested in finding the largest increase in gross sales. This kind of interestingness is based on user-defined utility functions in addition to the raw data [Shen et al., 2002; Cai et al., 1998; Hilderman et al., 1998; Wang et al., 2002; Yao et al., 2004; Chan et al., 2003; Lu et al., 2001; Zhang et al., 2004].

(8) Actionability / Applicability

A pattern is *actionable* (or *applicable*) in some domain if it enables decision making about future actions in the domain [Ling et al., 2002; Wang et al., 2002; Piatesky-Shapiro and Matheus, 1994]. Actionability is sometimes associated with a pattern selection strategy. For example, suppose we mine rules $AB \to X$ and $BC \to Y$ from a data set, and then we encounter some data including the itemset *ABC*. Both rules can be applied with different conclusions. The rule selection strategy determines which rule to apply, based perhaps on the statistical significance, accuracy, or conciseness of the rules.

The above-mentioned interestingness factors are sometimes correlated with rather than independent of one another. For example, actionability may be a good approximation for surprisingness and vice versa [Silberschatz and Tuzhilin, 1996], conciseness often coincides with generality, and generality conflicts with peculiarity.

The conciseness, generality, reliability, and peculiarity factors depend only on the data and the patterns, and thus can be considered as objective. The novelty, surprisingness, and actionability factors depend on the user who uses the patterns as well as the data and patterns themselves, and thus can be considered as subjective. The utility factor relates to the explanation of the discovered patterns, and thus can be considered semantics based.The above-mentioned interestingness factors are sometimes correlated with rather than independent to each other. For example, actionability may be a good approximation for surpriseingness [Silberschatz and Tuzhilin, 1996] and conciseness often coincides with generality.

2.3.2 Objective Interestingness Measures

According to whether users are involved, interestingness measures are classified as either objective or subjective. An ***objective interestingness measure*** depends only on the structure of the data and patterns extracted from it, such as support and confidence. An overview of the objective interestingness measures and their criteria can be found in [Hilderman and Hamilton, 2001; Sahar, 1999; Tan et al., 2002]. A ***subjective interestingness measure*** depends on the data structure as well as the prior knowledge and specific needs of the user.

Most of the existing objective interestingness measures are made up for either association rules or summaries. We review these two kinds of measures in this section.

2.3.2.1 Objective Interestingness Measures for Associations

Objective measures for association rules have been thoroughly studied by many researchers. Table 2.3 lists 38 common objective measures. In the table, *A* and *B* represent the antecedent and the consequent of a rule, respectively. *P*(*A*) denotes the probability of A, *P*(*B*|*A*) denotes the conditional probability of *B* given *A*. The association measures based on probability are usually functions of contingency tables. These measures originate from the different areas, such as statistics (correlation coefficient, Odds ratio, Yule's Q, and Yule's Y [Tan et al., 2002; Yao et al., 1999; Ohsaki et al., 2004]), information theory (J-measure and mutual information [Tan et al., 2002; Yao et al., 1999; Ohsaki et al., 2004]), and information retrieval (accuracy and sensitivity/recall [Lavra et al., 1999; Yao et al., 1999; Ohsaki et al., 2004]).

Table 2.3 Objective interestingness measures for association rules

Measure	Formula
Support	$P(AB)$
Confidence /Precision	$P(B \mid A)$, $\max(P(B \mid A), P(A \mid B))$
Coverage	$P(A)$
Accuracy	$P(AB) + P(\neg A \neg B)$
Lift or Interest	$P(B \mid A) / P(B)$ or $P(AB) / P(A)P(B)$
Leverage	$P(B \mid A) - P(A)P(B)$
Added Value / Change of Support	$P(B \mid A) - P(B)$, $\max(P(B \mid A) - P(B), P(A \mid B) - P(A))$
Relative Risk	$P(B \mid A) / P(B \mid \neg A)$
Jaccard	$P(AB) / (P(A) + P(B) - P(AB))$

Certainty Factor	$(P(B \mid A) - P(B))/(1 - P(B))$, $\max((P(B \mid A) - P(B))/(1 - P(B)),$ $(P(A \mid B) - P(A))/(1 - P(A)))$
Odds Ratio	$\frac{P(AB)P(\neg A \neg B)}{P(A \neg B)P(\neg BA)}$
Yule's Q	$\frac{P(AB)P(\neg A \neg B) - P(A \neg B)P(\neg AB)}{P(AB)P(\neg A \neg B) + P(A \neg B)P(\neg AB)}$
Yule's Y	$\frac{\sqrt{P(AB)P(\neg A \neg B)} - \sqrt{P(A \neg B)P(\neg AB)}}{\sqrt{P(AB)P(\neg A \neg B)} + \sqrt{P(A \neg B)P(\neg AB)}}$
Klosgen	$\sqrt{P(AB)}(P(B \mid A) - P(B))$, $\sqrt{P(AB)} \max(P(B \mid A) - P(B), P(A \mid B) - P(A))$
Brin's Conviction	$\frac{P(A)P(\neg B)}{P(A \neg B)}$, $\max(\frac{P(A)P(\neg B)}{P(A \neg B)}, \frac{P(B)P(\neg A)}{P(B \neg A)})$
Gray and Orlowska's Interestingness weighting Dependency (GOI-D)	$((\frac{P(AB))}{P(A)P(B)})^k - 1) * P(AB)^m$, k, m: Coefficients of dependency and generality, weighting the relative importance of the two factors..
Laplace Correction	$\frac{N(AB)+1}{N(A)+2}$, $\max(\frac{N(AB)+1}{N(A)+2}, \frac{N(AB)+1}{N(B)+2})$
J-Measure	$P(AB)\log(\frac{P(B \mid A)}{P(B)}) + P(A \neg B)\log(\frac{P(\neg B \mid A)}{P(\neg B)})$
Yao and Liu's One Way Support	$P(B \mid A) * \log_2 \frac{P(AB)}{P(A)P(B)}$
Yao and Liu's Two Way Support	$P(AB) * \log_2 \frac{P(AB)}{P(A)P(B)}$

Yao and Liu’s Two Way Support Variation	$P(AB)*\log_2 \frac{P(AB)}{P(A)P(B)} + P(A\neg B)*\log_2 \frac{P(A\neg B)}{P(A)P(\neg B)} + P(\neg AB)*\log_2 \frac{P(\neg AB)}{P(\neg A)P(B)} + P(\neg A\neg B)*\log_2 \frac{P(\neg A\neg B)}{P(\neg A)P(\neg B)}$
∅-Coefficient (Linear Correlation Coefficient)	$\frac{P(AB)-P(A)P(B)}{\sqrt{P(A)P(B)P(\neg A)P(\neg B)}}$
Piatetsky-Shapiro’s	$P(AB)-P(A)P(B)$
Cosine	$\frac{P(AB)}{\sqrt{P(A)P(B)}}$
Information Gain	$\log \frac{P(AB)}{P(A)P(B)}$
Sebag-Schoenauer	$\frac{P(AB)}{P(A\neg B)}$
Least Contradiction	$\frac{P(AB)-P(A\neg B)}{P(B)}$
Example and Counter Example Rate	$1-\frac{P(A\neg B)}{P(AB)}$

Given an association rule $A \rightarrow B$, there are two main interestingness factors for the rule. The first factor is the generality, denoted as *support* $P(AB)$ or *coverage* $P(A)$, which represents the statistical significance or the generality of the rule. The second factor is the association, which represents the correlation between A and B. Some researchers have suggested that a good interestingness measure should include both factors. For example, Tan et al. proposed the IS measure $IS = \sqrt{I \times Support}$, where $I = \frac{P(A,B)}{P(A)P(B)}$ is the ratio between the joint probability of two variables with respect to their expected probability under the independence assumption [Tan et al., 2000]. The IS measure is composed of two factors, the support and the *I* value that represents the association between *A* and *B*. Lavrac et al. proposed a weighted relative accuracy $WRAcc = P(A)(P(B \mid A) - P(B))$ [Lavrac et al., 1999]. This measure also combine the coverage $P(A)$ and the correlation between *B* and *A*. This measure is identical to the Piatetsky-Shapiro's measure $P(AB) - P(A)P(B)$ [Piatetsky-Shapiro, 1991]. Other measures involving these two factors include Yao and Liu's two way support [Yao et al., 1999], Jacard [Tan et al., 2002], Gray and Orlowska's Interestingness weighting dependency [Gray and Orlowska, 1998], and Klosgen [Klosgen 1996].

Tan et al. call the measures that include both factors *appropriate measures*. They argue that any *appropriate measure* could be used to rank the association rules. [Tan et al., 2000]. They also show that such measures behave similarly, especially when support is low.

Many objective measures have been proposed for different applications. To analyze these measures, some properties for the measures have been proposed. We classify these properties in two categories, the desirable properties and the circumstantial properties. The *desirable properties* are those must be satisfied in any cases. The *circumstantial properties* are those that should be satisfied in some applications, but should not be satisfied in others.

Piatesky-Shapiro [Piatetsky-Shapiro, 1991] proposed three principles that should be obeyed by any intuitive objective measure *F*.

P1. $F = 0$ if *A* and *B* are statistically independent; i.e., $P(AB) = P(A)\mathrm{P}(B)$.

P2. *F* monotonically increases with *P*(*AB*) when *P*(*A*) and *P*(*B*) remain the same.

P3. *F* monotonically decreases with *P*(*A*) (or *P*(*B*)) when *P*(*AB*) and *P*(*B*) (or *P*(*A*)) remain the same.

We classify principles P1 – P3 as desirable properties.

The first principle states that the association rules that occur by chance have zero interest value, i.e., they are not interesting. In practice, this principle may seem too rigid and some researches have proposed a constant value instead that F should have for independent situations [Tan et al., 2002]. The second principle states that the greater the support for *AB* is, the greater the interestingness value is when the support for *A* and *B* is fixed, i.e., the more positive correlation *A* and *B* have, the more interesting the rule is. The third principle states that if the supports for *AB* and *B* (or *A*) are fixed, the smaller the support for *A* (or *B*) is, the more interesting the pattern is. According to these two principles, when the cover of *A* and *B* are identical or the cover of *A* (*B*) contains the cover of *B* (*A*), the interestingness measure should attain the maximum value.

Tan et al. proposed five properties based on the operations on the contingency tables [Tan et al., 2002].

O1. *F* should be symmetric under variable permutation.

O2. *F* should be the same when we scale any row or column by a positive factor.

O3. *F* should become –*F* under row/column permutation, i.e., swapping the rows or columns in the contingency table should make the interestingness values change sign.

O4. *F* should remain the same under both row and column permutation.

O5. *F* should not depend on the count of the records that do not contain *A* and *B*.

Properties O1 – O4 can be considered as circumstantial. Property O1 states that rules $A \rightarrow B$ and $B \rightarrow A$ should have the same interestingness values, which is not true for many applications. Property O2 states that the invariance with the scaling the rows and/or columns. Many well know measures, such as Piatetsky-Shapiro's and Lift, do not satisfy this property. Property O3 states that $F(A \rightarrow B) = -F(A \rightarrow \text{not } B) = -F(\text{not } A \rightarrow B)$ which means that the measure can identify both positive and negative correlations. This property is not suitable for association rule mining where only positive correlation is of concern. Property O4 states that $F(A \rightarrow B) = F(\text{not } A \rightarrow \text{not } B)$. Property O3 is a special case of property O4. So it is not necessary for association rule mining either. Property O5 states that the measure only takes into account the number of the records containing *A* or *B* or both. Support does not satisfy this property while confidence satisfies it.

Lenca et al. [Lenca et al., 2004] proposed five properties to evaluate association rules.

Q1. *F* should be constant if there are no counterexamples to the rule.

Q2. $P(A \text{ not } B)$ should be linear concave or convex decrease around 0+.

Q3. Sensitivity to (increase with) the total number of records.

Q4. Ease of fixing a threshold.

Q5. Intelligibility, i.e., semantics can be easily expressed.

Properties Q1 – Q3 can be considered as circumstantial. Property Q1 states that rules with confidence of 1 should have the same interestingness value regardless of the support, which is not always desirable. Property Q2 states how dramatically the interestingness value decrease when there is only a few counterexamples. If the system can tolerate a few counterexamples, a concave decrease is desirable. If the system is strict about the confidence, a convex decrease is desirable. Property Q3 states that the measure should be sensitive to the size of the data set as well as the probabilities. Neither support nor confidence satisfies this property. Properties Q4 and Q5 can be considered as desirable, because these two properties define the friendliness of the measures to the users. A measure with semantics and easy to define threshold are easy to use.

2.3.2.2 Objective Interestingness Measures for Summaries

Diversity is an important concept that is used for measuring the interestingness of summaries. Although there is no consensus on the definition of diversity, researchers agree that the measures of diversity can be defined on the proportional distribution of the population and the number of classes [Hilderman and Hamilton, 2001]. Table 2.4 lists several measures for diversity [Hilderman and Hamilton, 2001]. In the definition, p_i denotes the probability for class *i*, $\bar{q} = 1/m$ denotes the average probability for all classes, n_i is the total count for class t_i, *N* is the total count for tuples in the table, *m* is the number of classes in the summary, $\bar{u} = N/m$ is the average count for each class, and $r_i = (n_i + \bar{u})/2N$ is the probability for class t_i according to the distribution *r*.

Table 2.4 Interestingness measures for summaries

Measure	Definition	P1	P2	P3	P4	P5
Variance	$\frac{\sum_{i=1}^{m}(p_i - \bar{q})^2}{m-1}$	√		√	√	√
Simpson	$\sum_{i=1}^{m} p_i^2$	√	√	√	√	√
Shannon	$-\sum_{i=1}^{m} p_i \log_2 p_i$				√	

Total	$-m\sum_{i=1}^{m} p_i \log_2 p_i$				√	
Max	$\log_2 m$				√	
McIntosh	$\frac{N-\sqrt{\sum_{i=1}^{m} n_i^2}}{N-\sqrt{N}}$				√	
Lorenz	$\bar{q}\sum_{i=1}^{m}(m-i+1)p_i$			√		
Gini	$\frac{\bar{q}\sum_{i=1}^{m}\sum_{j=1}^{m}\lvert p_i-p_j\rvert}{2}$	√	√		√	√
Berger	$\max(p_i)$	√	√	√	√	
Schutz	$\frac{\sum_{i=1}^{m}\lvert p_i-\bar{q}\rvert}{2m\bar{q}}$	√	√		√	
Bray	$\frac{\sum_{i=1}^{m}\min(n_i,\bar{u})}{N}$			√	√	
Whittaker	$1-\frac{\sum_{i=1}^{m}\lvert p_i-\bar{q}\rvert}{2}$			√	√	
Kullback	$\log_2 m-\sum_{i=1}^{m} p_i \log_2 \frac{p_i}{\bar{q}}$				√	
MacArthur	$-\sum_{i=1}^{m} r_i \log_2 r_i - \frac{\sum_{i=1}^{m} p_i \log_2 p_i + \log_2 m}{2}$	√	√		√	√
Theil	$\frac{\sum_{i=1}^{m}\lvert p_i \log_2 p_i - \bar{q}\log_2 \bar{q}\rvert}{m\bar{q}}$	√			√	
Atkinson	$1-\prod_{i=1}^{m}\frac{p_i}{\bar{q}}$	√	√		√	√

Although there is no single mathematical definition of diversity which has been widely accepted as standard, Hilderman and Hamilton proposed some general principles that a good measure should satisfy [Hilderman and Hamilton, 2001].

P1. Minimum Value Principle. Given a vector $(n_1, \ldots, n_m)$, where $n_i = n_j$ for all i, j, measure $f(n_1, \ldots, n_m)$ attains its minimum value.

This property indicates that the uniform distribution is the most uninteresting.

P2. Maximum Value Principle. Given a vector $(n_1, \ldots, n_m)$, where $n_1 = N - m + 1$, $n_i = 1$, $i = 2, \ldots, m$, and $N > m$, $f(n_1, \ldots, n_m)$ attains its maximum value.

This property shows that the most uneven distribution is the most interesting.

P3. Skewness principle. Given a vector $(n_1, \ldots, n_m)$, where $n_1 = N - m + 1$, $n_i = 1$, $i = 2, \ldots, m$, and $N > m$, and a vector $(n_1 - c, n_2, \ldots, n_m, n_{m+1}, n_{m+c})$, where $n_1 - c > 1$, $n_i = 1$, $i = 2, \ldots, m + c$, $f(n_1, \ldots, n_m) > f(n_1 - c, n_2, \ldots, n_m, n_{m+1}, \ldots, n_{m+c})$.

This property specifies that when the number of the total frequency remains the same, the interestingness measure for the most uneven distribution decreases when the number of the classes of tuples increases. This property has a bias for small number of classes.

P4. Permutation Invariance Principle. Given a vector $(n_1, \ldots, n_m)$ and any permutation $(i_1, \ldots, i_m)$ of $(1, \ldots, m)$, $f(n_1, \ldots, n_m) = f(n_{i1}, \ldots, n_{im})$.

This property specifies that interestingness for diversity has nothing to do with the order of the class; it is only determined by the distribution of the counts.

P5. Transfer principle. Given a vector $(n_1, \ldots, n_m)$ and $0 < c < n_j < n_i$, $f(n_1, \ldots, n_i + c, \ldots, n_j - c, \ldots, n_m) > f(n_1, \ldots, n_i, \ldots, n_j, \ldots, n_m)$.

This property specifies that when a positive transfer is made from the count of one tuple to another tuple whose count is greater, the interestingness increases.

2.3.3 Subjective Interestingness Measures

Although objective interestingness measures may capture the essence of the data, they are inadequate for many real applications, because (1) the domains of applications are different from each other, (2) the background knowledge of the different users is different, (3) the interest of the different users is different, and (4) the background knowledge of a user evolves. If an interestingness measure takes into account the background knowledge, expectation, or belief of the user, it is called a ***subjective interestingness measures***. Unlike the objective measures, subjective measures sometimes cannot be represented by mathematical formulas, because the user's knowledge can be represented in various forms. Instead, the user's knowledge is usually incorporated in mining procedures.

Liu et al. classify interestingness of discovered rules into three categories, finding unexpected patterns, conforming user's knowledge, and finding actionable patterns [Liu et al., 1997]. Unexpected patterns are the patterns that are unknown to users. Three types of unexpectedness of a rule are unexpected consequent, unexpected reason, and totally unexpected patterns. Conforming patterns are intended to validate the user's knowledge by the mined patterns. Actionable patterns can help the user do something to his/her advantage. For actionable patterns, the user should provide the situations under which he/she may take actions. They distinguish two types of user's knowledge, the ***general impressions*** (***GI***) and ***reasonably precise knowledge*** (***RPK***). GI represents a user's vague feelings. For example, the user may feel that annual income has relation with the chances of obtaining a loan, but he may not know the detailed correlation. RPK represents a user's more definite idea. For example, the use may believe that if his/her annual income is more than sixty thousand dollars, he/she will be granted a loan.

For the GI knowledge, Liu et al. proposed two specifications for defining the user's vague knowledge, T1 and T2 [Liu et al., 1997]. T1 can express the positive and negative relations between a condition variable and a class, the relation between a range (or a subset) of values of condition variables and class, and even vaguer impressions that there is a relation between a condition variable and a class. T2 extends T1 in that it separates user's knowledge into core and supplement. The core represents user's certain knowledge and the supplement represents user's uncertain knowledge. The core and a subset of supplement have a relation with a class. The authors then proposed several match algorithms for obtaining interestingness patterns for these two kinds of specifications for conforming rules, unexpected conclusion rules, and unexpected condition rules.

For the RPK knowledge, the authors represent the user's knowledge in the form of fuzzy rules. The system matches each discovered pattern against the fuzzy rules. The discovered patterns are then ranked according to their degrees of match. The authors proposed different matching algorithms for the three categories. All these interestingness measures are based on functions of fuzzy values that represent the match between the users knowledge and the discovered patterns.

Padmanabhan and Tuzhilin proposed another method to find unexpected rules based on user's knowledge [Padmanabhan and Tuzhilin, 1998]. In this method, the user's beliefs are represented in the same format as mined rules. Only rules that contradict with the beliefs are mined. The algorithm to find unexpected rules consists of two parts, ZoominUR and ZoomoutUR. For a belief $X \to Y$, ZoominUR attempts to find all exception rules of the form $X, A \to \neg Y$, i.e., all rules that are more specific than a belief but that have the contradictory consequence to that belief. Then ZoomoutUR generalizes the rules found by ZoominUR. For the rule $X, A \to \neg Y$, ZoomoutUR finds all rules $X', A \to \neg Y$, where X' is a subset of *X*, that are consistent with the data. Although

ZoominUR and ZoomoutUR use specifications for represent the user’s knowledge as GI and RPK, they have different mining process. GI and RPK methods work on a mined rule set and ranks the interestingness of the rules. ZoominUR and ZoomoutUR integrate the user’s expectations in the mining process to narrow down the mining space and they do not rank the rules.

To reduce the volume of work posted to the user in defining specifications for the user’s knowledge, Sahar proposed an interactive method to remove uninteresting rules without user’s specifications before the mining process [Sahar, 1999]. The method consists of three steps. First, the system selects the best candidate rule that has one condition attribute and one consequence attribute. The best candidate rule has the largest coverage, i.e., the number of the mined rules that contain the condition and consequence of the candidate rule is maximal. Second, the best candidate rule is presented for user to be classified into four categories, not-true-not-interesting, not-true-interesting, true-not-interesting, and true-and-interesting. If the rule is not-true-not-interesting or true-not interesting, the system removes the candidate rule and all rules it covers, i.e., all rules that contain the conditions and the consequences of the candidate rule. If a rule is not-true-interesting, the system removes this rule and all rules that it covers that have the same condition and keep all rules it covers that have additional condition attributes. Finally, if a rule is true-interesting, the system keeps it and its covered rules. After the best candidate rule is processed, the system selects the next best candidate rule and this process iterates until the rule set is empty or the user halts the process.

The advantage of this method is that users are not required to offer a specification of knowledge in advance; instead, they work with the system interactively. They only need to classify simple rules into true or false, interesting or uninteresting, and then the system may eliminate significant numbers of uninteresting rules. The drawback of this method is that it only makes the rule set smaller; it does not recommend which rules are the most interesting to users.

The above-mentioned subjective interestingness measures and methods are proposed for association rule mining. Silberschatz and Tuzhilin defined a subjective interestingness measure in a broader perspective. They relate the unexpectedness of discovered patterns to a belief system and define beliefs on arbitrary predicate formulae in first order logic [Silberschatz and Tuzhilin, 1996]. They classify beliefs into hard beliefs and soft beliefs. A ***hard belief*** is a constraint that cannot be changed with new evidence. If the evidence (patterns mined from data) contradicts hard beliefs, they assume that mistakes have been made in acquiring the evidence. A ***soft belief*** is a belief that the user is willing to change as new patterns are discovered that provide the user with new evidence. They assume that the degree of belief can be measured with conditional probability and adopt Bayesian approach to define the interestingness measure based on changes in soft beliefs. Given evidence *E* (a discovered pattern), the degree of a soft belief α is updated with the following Bayes rule

$$P(\alpha \mid E,\xi) = \frac{P(E \mid \alpha,\xi)P(\alpha \mid \xi)}{P(E \mid \alpha,\xi)P(\alpha \mid \xi) + P(E \mid \neg\alpha,\xi)P(\neg\alpha \mid \xi)},$$ where ξ is the context. Then, the interestingness measure for pattern *p* relative to a set of soft beliefs *B* is defined as the relative difference of the prior and posterior probability

$$I(p,B) = \sum_{\alpha\in B} \frac{|P(\alpha \mid p,\xi) - P(\alpha \mid \xi)|}{P(\alpha \mid \xi)}.$$

Although Silberschatz and Tuzhilin described their overall approach to subjective measures of interestingness, they did not provide a formal and detailed procedure and real applications. Another drawback is that it still needs to consult the original data to calculate the interestingness measure after the rules are mined.

CHAPTER 3

ESTIMATE PROPAGATION METHODOLOGY IN GENSPACE GRAPHS

In this chapter, we describe the framework for the GenSpace summary mining problem and propose the GSEP method for solving the GSSM problem. In Section 3.1, we describe the GSSM problem. In Section 3.2, we define the basic concepts relevant to the GSSM problem. In Section 3.3, we formalize the GSEP problem and propose a linear propagation method for solving this problem. In Section 3.4, we use an example to illustrate the GSEP process. In Section 3.5, we compare the linear GSEP method with Bayesian belief update and logic based belief revision.

3.1 An Overview of Summary Mining in GenSpace Graphs

In the beginning of the GSSM process, the user needs to specify his/her estimates at a certain aggregation level. These estimates need to be propagated in the whole graph for calculating the interestingness measure for the summaries. Generally speaking, the GSSM procedure consists of the following six steps [Geng and Hamilton, 2003b; Geng and Hamilton, 2004].

1. A domain generalization graph (DGG) for each attribute is created by explicitly identifying the domains appropriate to the relevant levels of granularity and the mappings between the values in these domains. The estimates (estimated probability distributions) are specified by the user for some nodes in each DGG to

form an ExGen graph and then propagated to all the other nodes in the ExGen graph.

2. The framework of the GenSpace graph is generated based on the ExGen graphs for individual attributes.
3. Aggregation is performed by transforming values in one domain to another, according to the directed arcs in the DGGs, and the potentially interesting nodes in this graph are materialized.
4. The given estimates are propagated throughout the GenSpace subgraph consisting of potentially interesting nodes (GSEP) [Hilderman and Hamilton, 2001].
5. The interestingness measures for these nodes are calculated and the highest ranked summaries are displayed (GSSM).
6. Estimates in the GenSpace graph are then adjusted and steps are repeated as necessary.

In the mining process, many issues need to be considered. In the GSEP process, we need to identify principles that any propagation method should embody, select an appropriate propagation method, and ensure the efficiency of the propagation method. In the GSSM process, we need to choose an appropriate interestingness measure. In the remainder of this chapter, we concentrate on the methodology for estimate propagation in GenSpace graphs.

3.2 Basic Concepts

In the propagation process, an ExGen graph is used to represent the user's knowledge relevant to generalization for a single attribute, while a GenSpace graph is used for multiple attributes. First, we give some formal definitions.

Definition 3.1 Given a set $X = \{x_1, x_2, \ldots, x_n\}$ representing the domain of some attribute and a set $\rho = \{\rho_1, \rho_2, \ldots, \rho_m\}$ of partitions of the set X, we define a nonempty binary relation $\preceq$ (called a ***generalization relation***) on ρ, where we say $\rho_i \preceq \rho_j$ if for every section $S_a \in \rho_i$, there exists a section $S_b \in \rho_j$, such that $S_a \subseteq S_b$. ρ_i is ***a finer partition*** of ρ_j. ρ_j is a ***rougher partition*** of ρ_i. For convenience, we often refer to the sections by labels.

If $\rho_i \preceq \rho_j$, for each section $S_b \in \rho_j$, there exists a set of sections $\{S_{a_1}, \ldots, S_{a_k}\} \subseteq \rho_i$, denoted ***Spec***($S_b$, ρ_i), such that $S_b = \bigcup_{i=1}^{k} S_{a_i}$. The generalization relation $\preceq$ is a partial order relation.

Example 3.1. Let *MMDDMAN* be a domain of morning, afternoon, and night of a specific non-leap year {*Morning of January* 1, *Afternoon of January* 1, *Night of January* 1, …, *Night of December* 31}, and ρ a set of partitions {*MMDD*, *MM*, *Week*, *MAN*}, where *MMDD* = {*January* 1, *January* 2, …, *December* 31}, MM = {*January*, *February*, …, *December*}, *Week* = {*Sunday*, *Monday*, …, *Saturday*}, and *MAN* = {*Morning*, *Afternoon*, *Night*}. Values of *MMDDMAN* are assigned to the values of the partitions in the obvious way, i.e., all *MMDDMAN* values that occur on *Sunday* are assigned to the *Sunday* value of *Week*, etc. Here *MMDD* $\preceq$ *MM* and *MMDD* $\preceq$ *Week.*

Definition 3.2 (adapted from [Hamilton et al., 2003; Hamilton et al., 2005]) A ***domain generalization graph*** (*DGG*) $G = \langle \rho, Arc \rangle$ is constructed based on a generalization relation $\langle \rho, \preceq \rangle$ as follows. The nodes of the graph are the elements of ρ. There is a directed arc from ρ_i to ρ_j iff $\rho_i \neq \rho_j$, $\rho_i \preceq \rho_j$, and there is no $\rho_k \in \rho$ such that $\rho_i \preceq \rho_k$ and $\rho_k \preceq \rho_j$. Each node corresponds to a domain of values called ***sections***. Each arc corresponds to a generalization relation, which is a mapping from the values in the domain of the initial (or ***parent***) node to that of the final node (or ***child***) of the arc. The ***bottom*** (or ***source***) node of the graph corresponds to the original domain of values *X* and the ***top*** (or ***sink***) node *T* corresponds to the most general domain of values, which contains only the value *Any*.

We call the more specific node the "parent node" and the more general node the "child node", because in the generalization process the directed arc goes from the specific node to the general node. Although this terminology may be counter intuitive for readers familiar with trees, it is normal for directed acyclic graphs.

From Example 3.1, we obtain the DGG shown in Figure 3.1.

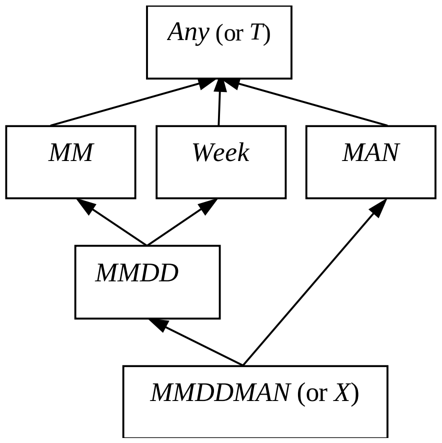


Figure 3.1 An Example DGG

In Figure 3.1, node *MMDDMAN* is the bottom node, representing the most specific level of domain. The *Any* node is the top node corresponding to most general level of the domain. Node *MMDD* is a parent of Node *MM*, because node *MMDD* can be generalized to node *MM*. Node *MM* has 12 sections, each of which represents a month, node *Week* has seven sections, each of which represents a day in a week, and so on.

Figure 3.1 shows that a DGG is a graphical structure that can be used both as a navigational aid by the user and as a guide to heuristic data mining procedures. The nodes in a DGG are domains of values for a given attribute, and the arcs tell how to generalize values from one domain to values in another. Each path in the graph corresponds to a generalization consistent with the specified generalization relations.

Domain knowledge about concept generalization is often important for the data mining process. The most common data structures for representing the concept generalization are concept hierarchies and ontologies. Essentially, a concept hierarchy is a tree, with the root node representing the most general concept and the leaf nodes representing the most specific concepts. Unfortunately, a concept hierarchy can represent only one generalization at a level. Since a concept hierarchy represents a totally ordered generalization relation, any concept hierarchy can be converted into a path in a DGG.

The term "ontology," which originated in philosophy, refers to the science of describing the kinds of entities in the world and how they are related. An ontology not only describes the classes, entities, properties, and concepts relevant to a type of knowledge, but also gives specifications for the relations and constraints among the entities. An ontology can represent sophisticated relations in a domain, but computational complexity is introduced if all aspects are applied during the data mining process. In work to date, simplified ontologies have been applied in data mining in two ways. First, ontologies have been used to customize data integration during information extraction from heterogeneous and distributed data sources [Silvescu et al., 2001]. Secondly, they have been used to mine knowledge at different levels of abstraction. In this case, the ontology was specified as a group of related concept hierarchies [Zhang et al., 2002], which is a less rich representation than a DGG.

Definition 3.3 An ***estimated distribution domain generalization*** (or ***ExGen***) ***graph*** $\langle \rho, Arc, E \rangle$ is a DGG that has estimates associated with every node. Estimates represent the estimated probability distribution of the occurrence of the sections in the domain corresponding to the node. For a node (i.e., partition) $\rho_j = \{S_1, \ldots, S_k\}$, we have $\forall S_i \in \rho_j,\ 0 \le E(S_i) \le 1$ and $\sum_{i=1}^{k} E(S_i) = 1$, where $E(S_i)$ denotes the estimated probability of the occurrence of section S_i.

An ExGen graph is a DGG where each node has been augmented with a description of the estimated distribution (or simply ***estimates***) of the values in the corresponding domain.

The estimate for a section S_i in node N is denoted as $E(S_i)$. The estimates for a node $N = \{S_1, S_2, \ldots, Sn\}$is denoted as $E(N) = [E(S_1), E(S_2), \ldots, E(Sn)]$. Continuing Example 3.1, for the node *MAN* = {*Morning*, *Afternoon*, *Night*}, we have *E*(*Morning*) = 0.2, *E*(*Afternoon*) = 0.5, *E*(*Night*) = 0.3, and *E*(*MAN*) = [0.2, 0.5, 0.3].

In the remainder of this thesis, we use "summary", "node", and "partition" interchangeably.

Definition 3.4 Assume node Q is a parent of node R in an ExGen graph, and therefore for each section $S_b \in R$, there exists a set of specific sections $Spec(S_b, Q) = \{ S_{a_1}, ..., S_{a_k} \} \subseteq Q$, such that $S_b = \bigcup_{i=1}^{k} S_{a_i}$. If for all $S_b \in R$, $E(S_b) = \sum_{i=1}^{k} E(S_{a_i})$, we say that nodes Q and R are ***consistent*** and vice versa.

For example, if estimates are evenly distributed at node *MMDD*, i.e., *E*(*MMDD*) = [1/365, 1/365, ……, 1/365], and E(*MM*) = [31/365, 28/365, 31/365, 30/365, 31/365, 30/365, 31/365, 31/365, 30/365, 31/365, 30/365, 31/365], we say nodes *MMDD* and *MM* are consistent.

According to the definition, the consistency relation is reflexive and symmetric. However, it is not transitive, as illustrated in Example 3.2.

Example 3.2. The domain for the ExGen graph is {*a*, *b*, *c*, *d*, *e*}, with the estimates uniformly distributed among all five elements. Nodes *P* and *R* are generalizations of node *X*. Node *Q* is a generalization of *R* and *P*. In this graph, *X* and *P* are consistent, *P* and *Q* are consistent, *Q* and *R* are consistent. However, *X* and *R* are not consistent, because for section *ab* of *R*, we have *E(ab*) = 0.2, which is not equal to the sum of sections *a* and *b* in of *X*, which have *E*(*a*) = 0.2 and *E*(*b*) = 0.2.

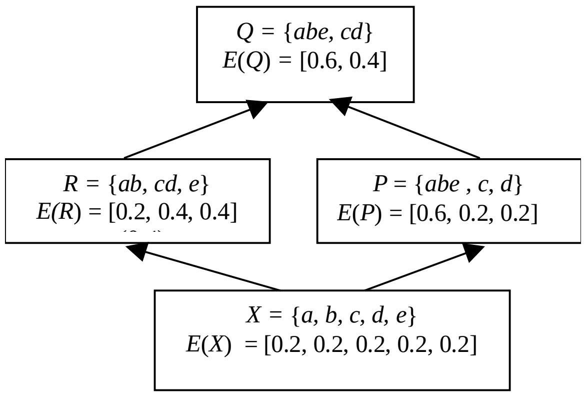


Figure 3.2 Consistency is not transitive

This example shows that we cannot use transitivity to infer consistency in an ExGen graphs. To make an ExGen graph consistent, we have to ensure that every pair of parent-child nodes is consistent.

Definition 3.5 An ExGen graph *G* is ***consistent*** if all pairs of adjacent nodes in *G* are consistent.

Definition 3.6 In an ExGen graph, we say that node *R* is ***bottom-consistent***, i.e., consistent with the bottom node *X*, if for all $S_i \in R$, $E(S_i) = \sum_{x \in X} x$.

The bottom node *X* is itself bottom-consistent.

Theorem 3.1 An ExGen graph *G* is consistent iff every node in *G* is bottom-consistent.

Proof.

(1) Suppose *G* is consistent.

Let the ***distance*** between node *R* and node *Q* is the minimum number of arcs that must be traversed to get from *R* to *Q*. By Definition 3.6, since *G* is consistent, the nodes at a distance of 1 from the bottom node (the child nodes of the bottom node) are bottom-consistent.

Now assume the nodes at a distance of *k* from the bottom nodes are bottom-consistent. Any node *R* at a distance of *k*+1 from the bottom node must be a child node of some node *Q* at a distance of *k* from the bottom node. Because G is consistent, *Q* and *R* are consistent. Therefore, for every $S_b \in R$, we have $Spec(S_b, Q) = \{S_{a_1}, ..., S_{a_k}\}$ and

$$E(S_b) = \sum_{i=1}^{k} E(S_{a_i}) .$$

We know $E(S_{a_i}) = \sum_{x \in S_{a_i}} E(x)$, since node *Q* is assumed to be bottom-consistent.

Hence, $E(S_b) = \sum_{i=1}^{k} E(S_{a_i}) = \sum_{i=1}^{k} \sum_{x \in S_{a_i}} E(x) = \sum_{x \in S_b} E(x)$, i.e., *R* is bottom-consistent.

(2). Suppose every node in *G* is bottom-consistent.

Let R be an arbitrary non-bottom node with parent Q. For any $S_b \in R$, we have $Spec(S_b, Q) = \{S_{a_1}, ..., S_{a_k}\}$ and $S_b = \bigcup S_{a_i}$. Since Q and R are both bottom-consistent, we have $E(S_b) = \sum_{x \in S_b} E(x)$ and $E(S_{a_i}) = \sum_{x \in S_{a_i}} E(x)$. Therefore, $\sum_{i=1}^{k} E(S_{a_i}) = \sum_{i=1}^{k} \sum_{x \in S_{a_i}} E(x) = \sum_{x \in S_b} E(x) = E(S_b)$. R and Q are consistent. Since Q and R are an arbitrary parent-child pair, we say G is consistent.

When we propagate estimates in a node to other nodes in the ExGen graph, we have two groups of propagation orders. The first group is to propagate the estimates of one node to its neighbours (parents and children) and then propagate them in the same way throughout the graph. The second group is to propagate the estimates to the bottom node first, and then use bottom up propagation to reach the other nodes in the graph. In the first group of propagation, when we propagate the estimates upward and then downward, there is no way to ensure the consistency of the graph. For example, in Figure 3.2, when we propagate estimates from node X to P to Q, and then from Q to R, no propagation method can ensure the consistency of node X with node P. The reason is that in the upward propagation from X to Q, there is information loss, which cannot be recovered without referring to node X in the downward propagation to from Q to R. However, in the second group of propagation, we can guarantee the consistency according to Theorem 3.1. Therefore, we will adopt the bottom up propagation for solving our problem.

Here we distinguish the propagation order and propagation method in this way. The *propagation order* is the order in which the nodes are traversed. For example, we may use minimal neighbour propagation, breadth first bottom up propagation, or minimal parent bottom up propagation. The *propagation method* is the method used to decide the estimate values for every node. It may include the optimized propagation and linear propagation, which are discussed in the next section. Different propagation methods produce different propagation results.

An ExGen graph is based on one attribute. If the generalization space consists of more than one attribute, we need to construct a GenSpace graph. Each node in a GenSpace is the Cartesian product of nodes the ExGen graphs of all the attributes. The number of the nodes in the generalization state space is $\prod_{i=1}^{n} l_i$, where n is the number of the attributes and l_i is the number of nodes in the ExGen graph for attribute i.

During the knowledge discovery process, the estimates at a particular node, which reflect a user's knowledge about the corresponding domain, can be updated. As well, estimates can be allowed to propagate to other nodes in the GenSpace graph; for example, if the user revises the estimate about the number of shows watched in the evening, then the estimate about the number of shows watched from 8:00PM to 9:00PM can be automatically adjusted.

We define a GenSpace graph as follows.

Definition 3.7 Given a set of attributes $\{A_1, A_2, \ldots A_n\}$, and an ExGen graph $G_i = <\rho_i, Arc_i, E_i>$ for each attribute A_i, a ***generalization space*** is defined as $<\rho, Arc, E>$, where $\rho = \rho_1 \times \rho_2 \times \ldots \times \rho_n$. For two nodes $Q = [\rho_{Q1}, \rho_{Q2}, \ldots, \rho_{Qn}]$ and $R = [\rho_{R1}, \rho_{R2}, \ldots, \rho_{Rn}] \in \rho$, where $\rho_{Qi} \in \rho_i$ and $\rho_{Ri} \in \rho_i$ denotes the partition of attribute A_i in nodes Q and R, respectively, if $\rho_{Qi} \preceq \rho_{Ri}$ for $1 \le i \le n$, we say $Q \preceq R$. There is a directed arc from Q to R in *Arc* iff $Q \neq R$, $Q \preceq R$, and there is no $O \in P$ such that $Q \preceq O$ and $O \preceq R$. The generalization relationship $\preceq$ is a partial order relation and $\langle \rho, \preceq \rangle$ defines a partially ordered set. For each node, we attach a set of estimates (estimated probability distribution of occurrence) for the values in the domain corresponding to the node. For a node (i.e., partition) $Q = \{S_1, \ldots, S_k\}$, we have $\forall S_i \in Q, 0 \leq E(S_i) \leq 1$ and $\sum_{i=1}^{k} E(S_i) = 1$, where $E(S_i)$ denotes the estimate of occurrence of section S_i. A graph constructed in this way is called a ***generalization space graph***, or ***GenSpace graph***, $\langle \rho, Arc, E \rangle$.

It can be seen that any GenSpace graph satisfies the following conditions. There is a directed arc from D_i to D_j iff $D_i \neq D_j$, $D_i \preceq D_j$, and there is no $D_k \in D$ such that $D_i \preceq D_k$ and $D_k \preceq D_j$. The top node (or sink node) corresponds to the most general domain of values, which consists of only the value [*Any*, *Any*, …, *Any*].

The following is the algorithm for constructing a GenSpace graph.

Input: a set of ExGen graphs for a set of attributes.

1. Select all the bottom nodes of ExGen graphs and create the bottom node of GenSpace graph by their Cartesian product. Then put the node in a queue.
2. Select the head of the queue and for each element of the tuple find the children in its corresponding ExGen graph and replace the element of the tuple with the child nodes and form a set of new nodes. For the nodes that are not yet in the GenSpace graph, insert these nodes as the current nodes' children in GenSpace graph and also insert these nodes in the queue, otherwise, connect parent-child nodes in the GenSpace graph. Delete the current node in the queue.

3. Repeat step 2 until the queue is empty.

The following example illustrates the construction of a GenSpace graph. Given the two ExGen graphs representing the generalization relations for attributes *Day* and *Address* in Figure 3.3, we generate the GenSpace graph in Figure 3.4. Each node in the GenSpace represents a joint probability distribution for the corresponding nodes in the ExGen graphs.

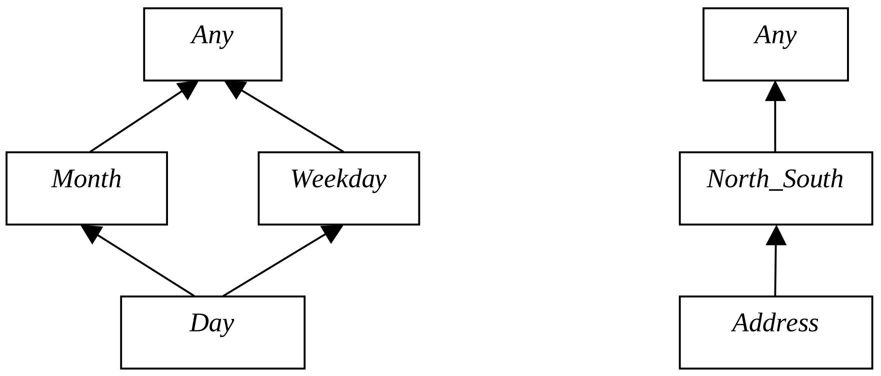


Figure 3.3. ExGens for single attributes

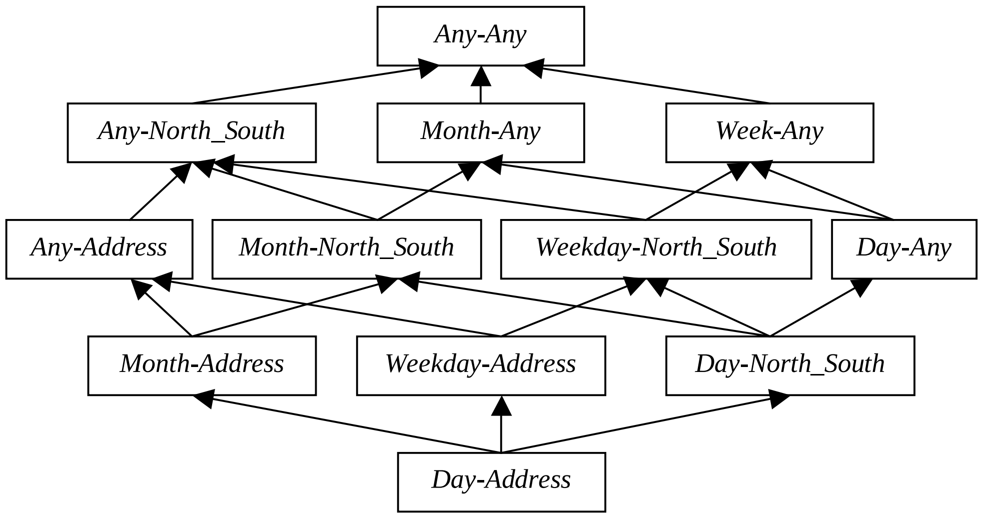


Figure 3.4 A GenSpace graph

Definitions 3.4, 3.5, and 3.6 and Theorem 3.1 for ExGen graphs can be adapted directly to GenSpace graphs. The only difference is that for an ExGen graph, the term “section” means a set of values for an attribute, while for a GenSpace graph, it means a set of tuples where each element in a tuple corresponds to a value for an attribute.

Corollary 3.1 A GenSpace graph *G* is consistent iff every node in *G* is bottom-consistent.

The structures of ExGen graphs and GenSpace graphs are directed acyclic graphs, but they are not necessarily lattices. However, a set of ExGen Graphs are all lattices, the structure of the GenSpace graph that results from combining them is also a lattice.

3.3 Propagation of Estimates in GenSpace Graphs

Due to the combinational number of the nodes in a GenSpace graph, it is not practical to require that the user specify the estimates for all nodes. In exploratory data mining, the user may begin with very little knowledge about a domain, perhaps only vague assumptions about the (a priori) probabilities of the possible values at some level of granularity. After the user specifies these estimates, a data mining system should be able to create default preliminary distributions for the other nodes.

With the GSSM system, the user can specify the estimates for each node in a GenSpace graph as an explicit probability distribution, or as a parameterized standard distribution, or leave the estimates unconstrained. If a standard distribution is specified, it is discretized into an explicit probability distribution in histogram form. If estimates are not specified for a node, they are obtained by propagation from a node with specified estimates.

In the last section, we discussed propagation orders and concluded that bottom up propagation preserves consistency. We adopt this propagation order, i.e., propagating estimates from a node to the bottom node, and then propagating upward to the other nodes. In bottom up propagation, the estimates are aggregated as in data cube calculation. The problem we face here is to determine the estimates in the bottom node according to the estimates specified at an arbitrary node. We also handle the case where estimates are specified at multiple nodes, but in this case we insist that the user not specify the distributions in two nodes that have generalization relation in a graph, because the information is either redundant or inconsistent.

In this propagation problem, we treat each estimate as either a ***hard constraint***, which once specified, must be satisfied after all subsequent propagations, or a ***soft constraint***, which must be satisfied for the current propagation, but thereafter need not be satisfied.

We propose the following three ***estimate propagation principles***: (1) propagations should be consistent, (2) if new estimates are being added, the new information should be preserved, while the old estimates should be changed as little as possible, and (3) if available information does not fully constrain the distribution at a node, the distribution should be made as even as possible.

The *minimal change* principle, which is the most common criterion in traditional accounts of belief change [Delgrande et al., 2004], states that a belief corpus should be modified in a minimal fashion when assimilating new information. The principle can take various forms [Makinson, 1993; Rott, 2000]. In our case, we first formalize it as an optimization problem and show how to solve this problem. We then propose a more efficient linear method as a heuristic solution for propagation.

3.3.1 Optimization Based Propagation

We identify six situations for propagating estimate: given estimates at only the bottom node, given no estimates for any node, given estimates at a single non-bottom node, given estimates at multiple nodes, and given updated estimates at one node. Let us consider each situation in turn.

(1) Given estimates at the bottom node (bottom-up propagation):

If the estimates at the bottom node are given, bottom up propagation preserves the consistency of the GenSpace graph regardless of the propagation path.

Assume the estimates at the bottom node X are given by $E(X) = [E(x_1), \ldots, E(x_n)]$, node Q is an arbitrary node that is consistent with X through upward propagation, R is a child node of Q, S_b is a section of R, and $Spec(S_b, Q) = \{S_{a_1}, ..., S_{a_k}\}$ is the set of the sections in Q that forms S_b. To propagate estimates from Q to R, we set the estimates for every section S_b of R to $\sum_{i=1}^{k} E(S_{a_i})$. Since $E(S_b) = \sum_{i=1}^{k} E(S_{a_i}) = \sum_{i=1}^{k} \sum_{x \in S_{a_i}} E(x) = \sum_{x \in S_b} E(x)$ for every $S_b \in R$, then R is bottom consistent. Therefore, bottom-up propagation in a GenSpace graph preserves consistency.

(2) Given no information about estimates in any node:

We assume that the estimates for the elements of the domain X are the same. In other words, we assume a uniform distribution at the bottom node X. Then we apply bottom-up propagation.

(3) Given estimates at a single non-bottom node:

Given estimates $E(R) = [E_1, \ldots, E_k]$ for a node $R = \{S_1, \ldots, S_k\}$, we distribute the estimates E_i uniformly for each section S_i among the sections in $Spec(S_i, X)$, and therefore obtain the estimate for the jth element of the ith section $E_{ij} = E_i / |Spec(S_i, X)|$. After obtaining the estimates at the bottom node X, we use bottom-up propagation.

(4) Given estimates in multiple nodes:

We find the greatest lower bound of all nodes whose estimates are specified. We first calculate the estimates at the greatest lower bound by solving a linear equation group, and then use the same method as with case 3 to determine the bottom node's estimates. The linear equation group is described as follows.

Given a set of nodes with estimates, the estimates at the greatest lower bound G of all these nodes is found by representing the constraint due to each node as an equation and then solving the set of equations. For a node i with j_i sections S_{ik}, where $1 \le k \le j_i$, we assume that E_{ik} is the estimate for section S_{ik}. We also assume that S_G is a section of G, i.e., $S_G \in G$. We have $E(S_{ik}) = E_{ik}$, i.e., $\sum_{S_G \in Spec(S_{ik}, G)} E(S_G) = E_{ik}$. There are three possible cases for the solutions of the linear equation group.

First, if there is exactly one non-negative solution, we assign the solution as the initial estimates at the greatest lower bound G and perform upward propagation.

Secondly, if more than one non-negative solution exists, we convert the problem to an optimization problem. To satisfy the equation group and at the same time to make the distribution as even as possible, we minimize the following expression:

$$\sum_{S_G \in G} (E(S_G) - 1/|G|)^2 .$$

Minimizing this expression is a constrained linear least-square problem, which is NP complete. Many iterative algorithms, called ***optimization methods***, have been proposed for approximating the solution to this problem [Schittkowski, 1985; Fletcher, 1987]. After obtaining the estimates at node *G* by solving this problem, and then determining the estimates at the bottom node, we use bottom-up propagation to determine the estimates for all other nodes.

Thirdly, if no non-negative solution exists, we request that the user change the input knowledge to remove the inconsistency.

(5) Given revised estimates for one node:

We need to satisfy the new conditions as well as all existing hard constraints, and at the same time minimize $\sum_{i=1}^{n}(E\_new(x)-E\_old(x))^2$, where *n* is the cardinality of the bottom node *X*, to make the changes at node *X* as small as possible. This problem is also a constrained linear least-square problem.

(6) Given revised estimates for multiple nodes:

We deal with this situation the same way as with situation 5 except that we combine the linear constraints obtained from all nodes whose estimates are revised.

3.3.2 Linear Propagation

For large scale GenSpace graphs, the above mentioned optimization process has a prohibitive time cost. We propose the ***linear propagation method*** to find a heuristic solution to the optimization problem. If $E\_old(S_{ik})>0$, we convert $E(S_{ik}) = E_{ik}$ to $E(x)=\frac{E\_old(x)}{E\_old(S_{ik})}E_{ik}, x\in S_{ik}$, which means that we accept the probability ratio among the groups and also retain the ratio among the elements of each group. If $E\_old(S_{ik}) = 0$, we convert $E(S_{ik}) = E_{ik}$ to $E(x) = \frac{E_{ik}}{|S_{ik}|}, x \in S_{ik}$, which means that we distribute the estimates evenly among the elements. Once new estimates come in a node, this propagation will be performed. Therefore, it is an iterative process.

This method has three advantages. First, it is computationally efficient, because it involves only linear computation with time complexity of O(|N||X|), where |N| is the size of node N where the user changed his/her estimates. Secondly, it does not need to resolve conflicts among the user's new estimates, because we propagate estimates in different nodes in a sequence rather than simultaneously as with an optimization method. Thirdly, as proved below in Lemma 3.1, we have identified an upper bound for the changes of estimates at any node in the GenSpace graph, given the change specified by the user at an arbitrary node. In this sense, the user's old information is preserved. The disadvantage is that all previously specified estimates are treated as soft constraints, i.e., after specifying and propagating a new constraint, any old constraint may be violated.

To identify the upper bound of the estimate change, we first define the ***relative variance*** as the measure for changes of estimates.

Definition 3.8 The ***relative variance of an estimate for a section*** S is defined as $rvs(S) = \frac{|E'(S) - E(S)|}{E(S)})$, where $E'(S)$ and $E(S)$ denote the old and new estimates, respectively.

Definition 3.9 The ***relative variance of estimates for a node*** $N = \{S_1, \ldots S_n\}$ is defined as $rvn(N) = \frac{1}{n}\sum_{i=1}^{n} rvs(S_i)$.

Lemma 3.1 Suppose node *N* has *m* sections S_1, ..., S_m and the bottom node *X* has *n* elements s_1, …, s_n, where $m \le n$. Let the old estimates for node *N* be *E*(*N*) = [$E(S_1)$, …, $E(S_m)$], for *X* be *E*(*X*) = [$E(s_1)$, …, $E(s_n)$]. Let the new estimates for *N* be $E'(N) = [E'(S_1), ..., E'(S_m)]$. Let $\alpha = \max_{i=1}^{m}(rvs(S_i))$. After propagating E' from *N* to *X* using the linear propagation method, $rvs(s)$ for any element *s* in the bottom node satisfies $rvs(s) \le \alpha$ and the relative variance *rvn*(*X*) at the bottom node satisfies $rvn(X) \le \alpha$.

Proof: For any element *s* in the bottom node, there exists a section *S* in *N* such that $s = Spec(S, X)$. Since we have $\frac{E'(S_i)}{E(S_i)} = \frac{E'(s)}{E(s)}$ according to the property of linear propagation,

$$rvs(s) = \left|\frac{E'(s) - E(s)}{E(s)}\right| = \left|\frac{E'(S) - E(S)}{E(S)}\right| \le \alpha .$$

Since for any s_i we have $rvs(s_i) \le \alpha$, we have $rvn(X) = \frac{1}{n}\sum_{i=1}^{n} rvs(s_i) \le \alpha$.

Lemma 3.2 Suppose node *N* has *m* sections S_1, ..., S_m and the bottom node *X* has *n* elements s_1, …, s_n, where $m \le n$. Let the old estimates for node *N* be *E*(*N*) = [$E(S_1)$, …, $E(S_m)$], for *X* be *E*(*X*) = [$E(s_1)$, …, $E(s_n)$]. Let the new estimates for *X* be $E'(X) = [E'(s_1), ..., E'(s_m)]$. Let $\alpha = \max_{i=1}^{n}(rvs(s_i))$. After propagating $E'(X)$ from *X* to *N*, $rvs(S)$ for any element *s* in any node *N* satisfies $rvs(s) \le \alpha$ and the relative variance *rvn*(*N*) satisfies $rvn(N) \le \alpha$.

Proof: For any section *S* in node *N*, there exists a set of elements *Spec*(*S*, *X*) corresponding to *S*.

$$rvs(S) = \left| \frac{E'(S) - E(S)}{E(S)} \right|$$
$$= \left| \frac{\sum_{s \in Spec(S,X)} E'(s) - \sum_{s \in Spec(S,X)} E(s)}{\sum_{s \in Spec(S,X)} E(s)} \right|$$
$$= \left| \frac{\sum_{s \in Spec(S,X)} (E'(s) - E(s))}{\sum_{s \in Spec(S,X)} E(s)} \right|$$
$$\leq \frac{\sum_{s \in Spec(S,X)} |E'(s) - E(s)|}{\sum_{s \in Spec(S,X)} |E(s)|}$$

Since for any s we have $\frac{|E'(s) - E(s)|}{E(s)} \leq \alpha$, or equivalently $|E'(s) - E(s)| \leq E(s)\alpha$,

therefore, $rvs(S) \leq \frac{\sum_{s \in Spec(S)} E(s)\alpha}{\sum_{s \in Spec(S)} |E(s)|} = \alpha$. Accordingly, we have $rvn(N) = \frac{1}{n}\sum_{i=1}^{m} rvs(S_i) \leq \alpha$.

Theorem 3.2 Suppose node *N* has *m* sections S_1, ..., S_m. Let the old estimates for node *N* be $E(N) = [E(S_1), \ldots, E(S_m)]$, the new estimates for *N* be $E'(N) = [E'(S_1), ..., E'(S_m)]$. Let $\alpha = \max_{i=1}^{m}(rvs(S_i))$. After propagating E' from *N* to the entire GenSpace graph using the linear propagation method, the relative variance $rvs(s)$ for any element *s* in any node *M* satisfies $rvs(s) \leq \alpha$ and the relative variance *rvn*(*M*) of node *M* satisfies $rvn(M) \leq \alpha$.

Proof: The proof follows directly from Lemmas 3.1 and 3.2.

Next, we identify independence properties that ensure that old constraints remain valid after changing estimates and that changing estimate at one node only affects a subset of nodes in the GenSpace graph.

Definition 3.10 For a pair of nodes, A = {S_{A1}, …, S_{Aa}} and B = {S_{B1}, …, S_{Bb}} in an ExGen / GenSpace graph, if $E(S_{Ai})E(S_{Bj}) = E(S_{Ai} \cap S_{Bj})$, for all i and j, we say that nodes A and B are independent.

Lemma 3.3 If a pair of nodes A = {S_{A1}, …, S_{Aa}} and B = {S_{B1}, …, S_{Bb}} in a GenSpace graph are independent, and we use the linear propagation method, then changing estimates at one node does not change the estimates in the other.

Proof

Assume we change the distribution of node B from $E(S_{B1})$, …, $E(S_{Bn})$ to $E'(S_{B1})$, …, $E'(S_{Bn})$.

Because $E'(S_{Ai}) = \sum_{j=1}^{n} E'(S_{Bj} \cap S_{Ai})$ and we use the linear propagation method, we have

$E'(S_{Bj} \cap S_{Ai}) = \dfrac{E'(S_{Bj})}{E(S_{Bj})} E(S_{Bj} \cap S_{Ai})$, and accordingly we have

$$E'(S_{Ai}) = \sum_{j=1}^{n} E'(S_{Bj} \cap S_{Ai}) = \sum_{j=1}^{n} \frac{E'(S_{Bj})}{E(S_{Bj})} E(S_{Bj} \cap S_{Ai}) .$$

Since A and B are currently independent, we have $E(S_{Ai}) = \dfrac{E(S_{Bj} \cap S_{Ai})}{E(S_{Bj})}$, therefore,

$$E'(S_{Ai}) = \sum_{j=1}^{n} E(S_{Ai})E'(S_{Bj}) = E(S_{Ai}) .$$

Lemma 3.4 If nodes A and B are independent, then A is independent of B's descendents, and vice versa.

Proof:

We have $E(S_{Ai}) * E(S_{Bj}) = E(S_{Ai} \cap S_{Bj})$ for all i and j, i = 1 to m, and j = 1 to n.

Assume C is a descendent node of A with k sections S_{Cm}, $1 \le m \le k$. We have

$$E(S_{cm}) * E(S_{Bj}) = \sum_{S_{Ai} \in Spec(S_{Pj}, A)} E(S_{Ai}) * E(S_{Bj})$$

$$= \sum_{S_{Ai} \in Spec(S_{Pj}, A)} E(S_{Ai} \cap S_{Bj}) = E(\bigcup_{S_{Ai} \in Spec(S_{Pj}, A)} S_{Ai} \cap S_{Bj}) = E(C_m \cap S_{Bj}).$$

We can infer that there is only one common descendent node, i.e., the top node, for any pair of independent nodes.

Definition 3.11 For a pair of nodes A = {A_1, …A_m} and B = {B_1, …B_n} in an ExGen/GenSpace graph, if $|A_i| * |B_j| = |A_i \cap B_j| * N$, where N denotes cardinality of the domain, holds for all *i* and *j*, *i* = 1 to *m*, and *j* = 1 to *n* we say *A* and *B* are structural independent.

Lemma 3.5 If *A* and *B* are structurally independent, then *A* is structurally independent to *B*'s children, and *B* is structurally independent to *A*'s children.

Proof:

We have $|A_i| * |B_j| = |A_i \cap B_j| * N$ for all *i* and *j*, *i* = 1 to *m*, and *j* = 1 to *n*.

Assume *C* is a child of *A* and a partition $C_m = A_{i_1} \cup ... \cup A_{i_k}$, where *k* is the number of partions in *A* that belong to C_m , we have

$$|C_m| * |B_j| = \sum |A_{i_k}| * |B_j| = \sum |A_{i_k} \cap B_j| * N = |\bigcup A_{i_k} \cap B_j| * N = |C_m \cap B_j| * N .$$

Lemma 3.6 If A = {A_1, …A_m} and B = {B_1, …B_n} are structurally independent and the estimates for the bottom node are uniformly distributed, then *A* and *B* are independent.

Note that nodes *A* and *B* are currently independent, but after update on the third node *C*, *A* and *B* are not necessarily be independent again.

From the results, we infer that if all paths between the bottom node and top node do not cross each other and all pairs of the children of the bottom node are independent, then to update the estimates in any node, we only need to propagate estimates along the path containing the node.

3.4 A Propagation Example

For simplicity, we use an ExGen graph to illustrate our propagation method. We use Example 3.1 and Figure 3.1 to illustrate the propagation methods presented in Section 3.1 and Section 3.2. We specify the estimates of accessing the computer system at the University of Regina for some nodes as follows: *E*(*MM*) = [0.09, 0.12, 0.12, 0.09, 0.04, 0.04, 0.04, 0.04, 0.09, 0.12, 0.12, 0.09], *E*(*Week*) = [0.1, 0.2, 0.2, 0.2, 0.1, 0.1, 0.1], and no information about the estimates of other nodes. Since the node *MMDD* is the greatest lower bound of *MM* and *Week*, we first calculate the estimates in *MMDD* for a non-leap year as *E*(*MMDD*) = [e_1, e_2, …, e_{365}]. Assuming January 1 is Sunday, we have the following linear constraints,

$$\sum_{i=1}^{31} e_i = 0.09\ ,\ \sum_{i=32}^{59} e_i = 0.12\ ,\ \cdots,\ \sum_{i=335}^{365} e_i = 0.09\ ,$$

$$\sum_{i\%7=1} e_i = 0.1\ ,\ \sum_{i\%7=2} e_i = 0.2\ ,\ \cdots,\ \sum_{i\%7=0} e_i = 0.1 \cdot$$

Then we minimize the following formula,

$$\sum_{i=1}^{365} (e_i - 1/365)^2 \cdot$$

We obtain *E*(*MMDD*) = [0.0020, 0.0040, 0.0040, …, 0.0022, 0.0021]. Next we calculate the estimates for the bottom node by dividing the estimates of *MMDD* by 3 (divide evenly among *Morning*, *Afternoon* and *Night*). We get estimates for the bottom node *E*(*MMDDMAN*) = [0.0007, 0.0013, …, 0.0007]. Finally, we use upward propagation and we obtain *E*(*MAN*) = [0.3333, 0.3333, 0.3333].

Now if new information is obtained that *E*(*MAN*) = [0.3, 0.5, 0.2], we have the following constraints at the bottom node.

$$\sum_{x \in Spec(Morning,\ MMDDMAN)} E(x) = 0.3,$$

$$\sum_{x \in Spec(Afternoon,\ MMDDMAN)} E(x) = 0.5,$$

$$\sum_{x \in Spec(Night,\ MMDDMAN)} E(x) = 0.2 \cdot$$

We minimize the formula $\sum_{x \in MMDDMAN} (E(x) - E\_old(x))^2$ and get the new estimates for the bottom node. Then after upward propagation, we get *E*(*MMDDMAN*) = [0.0006, 0.0011, …, 0.0003], *E*(*MM*) = [0.0884, 0.1186, 0.1184, 0.0885, 0.0433, 0.0427, 0.0431, 0.0432, 0.089, 0.1184, 0.1185, 0.0884] and *E*(*Weekday*) = [0.1021, 0.1974, 0.1974, 0.1974, 0.1021, 0.1019, 0.1018]. Thus, after adopting the new estimates in node *MAN*, the estimates at the other nodes are changed only a little, which means that the original information is significantly preserved.

If we specify all the constraints from *E*(*MMDD*) and *E*(*MAN*) at the same time, we obtain *E*(*MMDDMAN*) = [0.0006, 0.0011, …, 0.0003] and *E*(*MMDD*) = [0.0019, 0.0041, 0.0041, 0.0040, …, 0.0021, 0.0021]. The estimates for *MM* and *Weekday* and *MAN* are identical to the estimates specified for them.

If we use the linear propagation method, after all propagations, we obtain *E*(*MMDDMAN*) = [0.0006, 0.0010, …, 0.0004], *E*(*MM*) = [0.0938, 0.1111, 0.1222, 0.0875, 0.0426, 0.0389, 0.0407, 0.0417, 0.0875, 0.1250, 0.1195, 0.0895], *E*(*Weekday*) = [0.1000, 0.2000, 0.2000, 0.2000, 0.1000, 0.1000, 0.1000] and *E*(*MAN*) = [0.3000, 0.5000, 0.2000]. Figures 3.5 to 3.7 compare the estimates for the *Month*, *Weekday*, and *MAN* nodes obtained from the optimization method and the linear method. In Figures 3.5, the one-step optimization propagation method preserves the identical estimates for *Month* as the ones given by the user. The two-step optimization propagation method and the linear propagation method give very close results. In Figure 3.6, similar results are also observed. In Figure 3.7, the three methods produce the same results. Figures 3.8 and 3.9 compare the results for *MMDD* node for January and February. In general, the results obtained from linear propagation method are close to those obtained from optimization-based propagation.

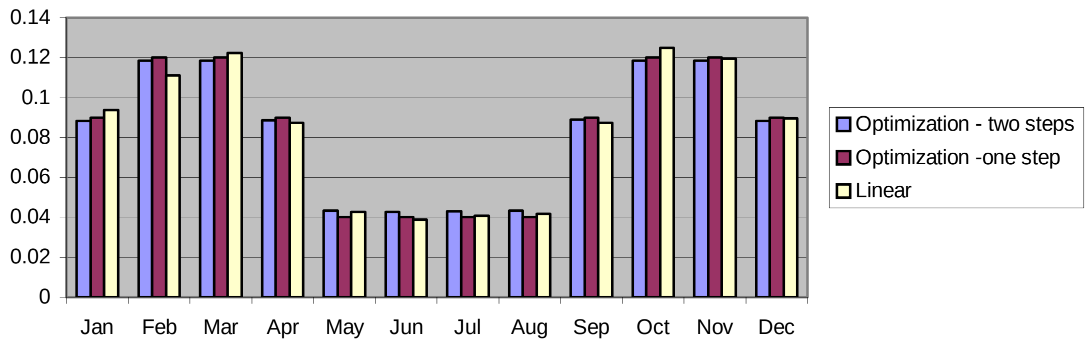


Figure 3.5 Estimated distributions for *Month*

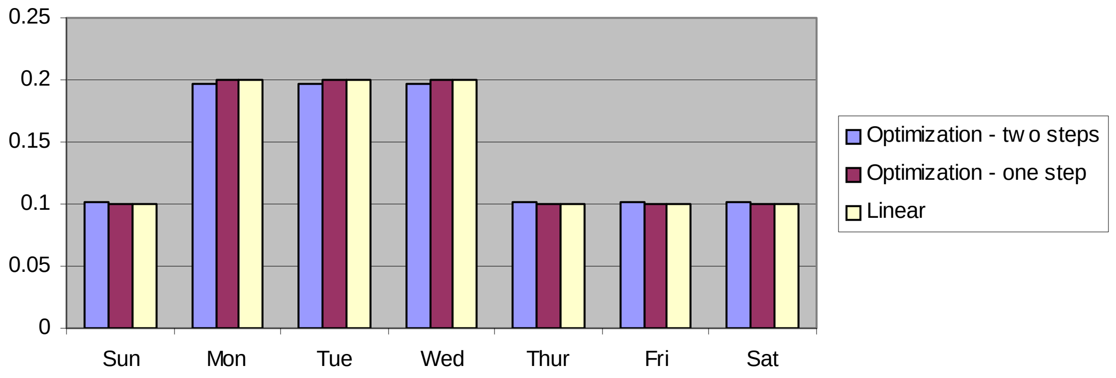


Figure 3.6 Estimated distributions for *Weekday*

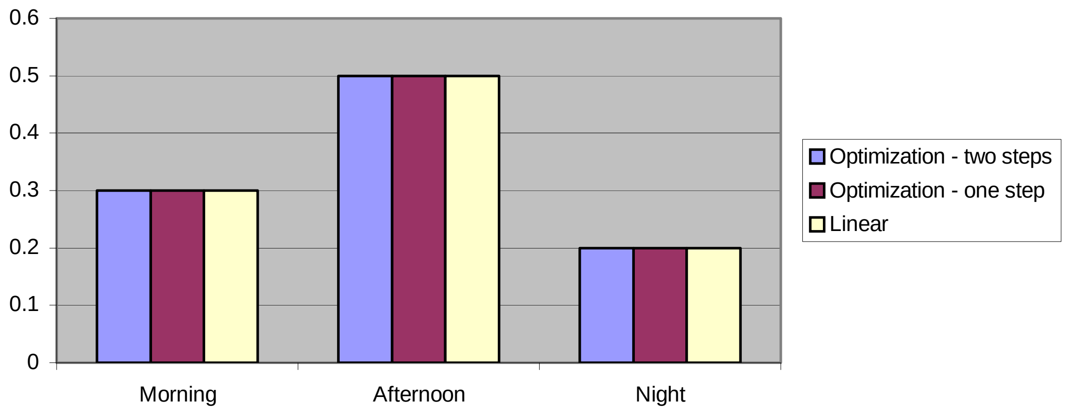


Figure 3.7 Estimated distributions for *MAN*

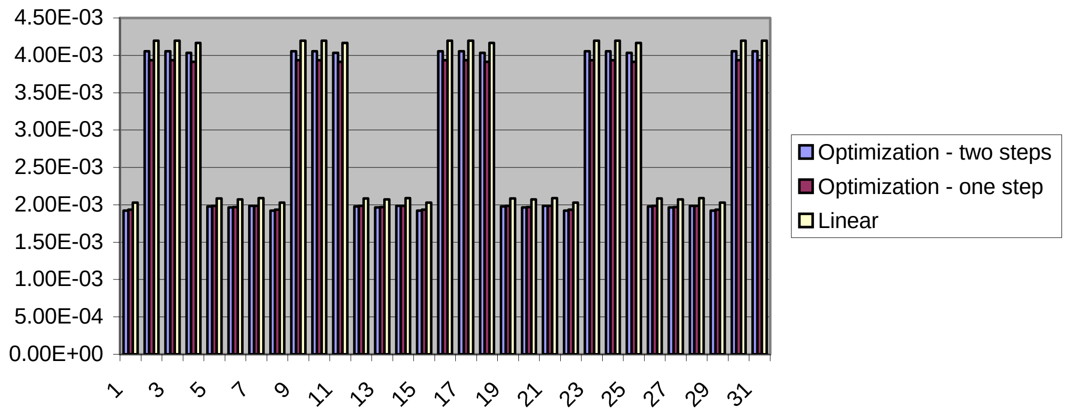


Figure 3.8 Estimated distributions for *MMDD* for *January*

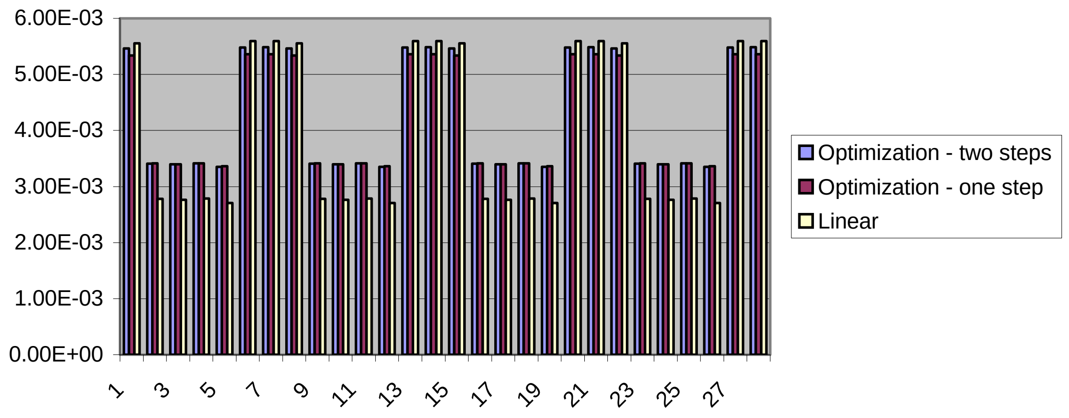


Figure 3.9 Estimated distributions for *MMDD* for *February*

Table 3.1 compares the running time of the optimization method and the linear method. The experiments were conducted using Matlab 6.1 on a PC with 512MB memory and 1.7 GHz CPU. First, we applied the optimization method twice. The total propagation time is 2051 seconds (67 seconds for the first propagation and 1984 seconds for the second). Then, we combined all the constraints and propagated the estimates with the optimization method. The running time was 4791 seconds. For the linear method, the time was imperceptible for all three propagations.

Table 3.1 Comparison of running time between optimization and linear methods

Method		# of variables	# of constraints	Running time (Sec)
Optimization - two steps	Step 1	365	18	67
	Step 2	1095	3	1984
Optimization - one step		1095	21	4791
Linear		1095		<1

3.5 Relation to Bayesian Belief Updating

Bayesian networks are models that use probability theory to manage uncertainty by explicitly representing the conditional dependencies between the attributes to improve the reasoning efficiency. They also provide an intuitive graphical visualization of a user's knowledge.

We identify five differences between Bayesian belief updating and linear GSEP. The most prominent difference between linear GSEP and Bayesian belief updating is that linear GSEP is a belief revision problem and Bayesian belief updating is a belief updating problem. A belief revision problem is a problem where the world is static and the user's knowledge about the world is not perfect and needs to be revised when new evidence arises. A belief updating problem is a problem where user's knowledge is assumed to be correct. When the new event happens, the status of the world changes and the user's knowledge should be updated to reflect the new circumstance.

Secondly, in Bayesian belief updating, the users know what domain they are interested in before propagation, and hence only that variable needs to be updated. In GSEP, users do not know what generalization levels they are interested in, and therefore, all nodes need to be updated.

Thirdly, linear GSEP and Bayesian belief updating use different methods to tackle the intractability problems in probability propagation. Jensen identified three intractability problems in Bayesian belief updating: acquisition intractability, updating intractability, and magnetization intractability [Jensen, 1996]. Bayesian belief updating uses conditional independence to tackle the intractability problems. In GSEP, we use a variety of methods to deal with these problems. To avoid acquisition intractability, we ask the user to specify estimates at high conceptual levels in the GenSpace graph, which contain tractable numbers of probabilities. To avoid updating intractability, we only take into consideration the tuples present in the original table. Finally, to avoid marginalization intractability (or generalization intractability, in our case), we adopt OLAP aggregation methods to tackle the efficiency problem.

Fourthly, Bayesian belief updating assumes each variable has only a single conceptual level. Inferences are performed on different variables and do not deal with multiple conceptual levels of a single attribute.

Fifthly, evidence in linear GSEP and Bayesian belief updating is of a different form. In Bayesian belief updating, the evidence can be considered as a probability distribution $[P_1, P_2, \ldots P_n]$. If event i happens, we have $P_i = 1$ and $P_j = 0$, where $j \neq i$. In

GSEP, the new estimates are in the form [P_1, P_2,… P_n], where $0 \le P_i \le 1$, $\sum_{i=1}^{n} P_i = 1$. In this sense, Bayesian belief updating can be considered as a special case of GenSpace propagation.

Linear GSEP is similar to Bayesian belief updating [Jensen, 1996; Xiang, 2002] in that (1) they are both based on probability theory; (2) they both propagate the user's beliefs or estimates from one domain to the others; (3) they both use linear calculation (normalization and marginalization) to proportionally propagate beliefs; (4) they both face the problems of acquisition intractability, updating intractability, and marginalization intractability [Jensen, 1996].

We use an example to demonstrate the relationship between linear GSEP and Bayesian belief updating. Therefore, they deal with different problems.

Example 3.3. Suppose we have four variables, *Season* (*S*), *City* (*C*), *Fishing* (*F*), and *Temperature* (*T*), with domains {*Summer*, *Other*}, {*Victoria*, *Montreal*}, {*Suitable*, *Unsuitable*}, and {*High*, *Low*}, respectively. We assume the Bayesian network shown in Figure 3.10 has been constructed for these variables. We assume that the prior probabilities for season and city are *P*(*S*) = {0.2, 0.8} and *P*(*C*) = {0.1, 0.9}. The remaining conditional probabilities are listed in Table 3.2.

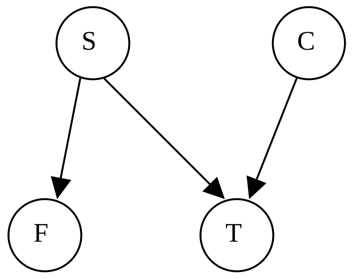


Figure 3.10 An example of Bayesian network

Table 3.2 The conditional probabilities for Example 3.3

P(F|S)	S = Summer	S = Other
F = Suitable	1	0.2
F = Unsuitable	0	0.8

P(T|S,C)	Summer, Victoria	Summer, Montreal	Other, Victoria	Other, Montreal
T = High	1	1	0.9	0
T = Low	0	0	0.1	1

Now, we calculate the prior probabilities for *F* and *T*.

Using formula $P(S, F) = P(F|S)\mathrm{P}(S)$, we have the probabilities for $P(F, S)$ as shown in Table 3.3.

Table 3.3 Probability distribution for $P(F, S)$

P(*F*, *S*)	*S = Summer*	*S = Other*
F = Suitable	0.2	0.16
F = Unsuitable	0	0.64

Marginalizing *S* out, we have $P(F) = \{0.36, 0.64\}$, which means that 36% of days are suitable for fishing.

Similarly, using the formula $P(S, C, T) = P(T \mid S, C)P(S)P(C)$, we calculate the probability distribution for $P(S, C, T)$ as shown in Table 3.4.

Table 3.4 Probability distribution for $P(S, C, T)$

P(T, S, C)	Summer, Victoria	Summer, Montreal	Other, Victoria	Other, Montreal
T = High	0.02	0.18	0.072	0
T = Low	0	0	0.008	0.72

Marginalizing *S* and *C* out of $P(T, S, C)$ yields $P(H) = (0.272, 0.728)$.

Now, suppose that we have the evidence T = *High* and want to obtain the updated probabilities $P^*(S)$, $P^*(C)$, and $P^*(F)$.

In Table 3.4, we set all entries with T = *Low* to zero and normalize the table by dividing by the sum of the remaining entries as shown in Table 3.5

Table 3.5 Probability distribution for $P^*(S, C, T)$ with evidence T = *High*

$P^*(T, S, C)$	Summer, Victoria	Summer, Montreal	Other, Victoria	Other, Montreal
T = High	0.074	0.662	0.264	0
T = Low	0	0	0	0

By marginalization, we have $P^*(S) = (0.736, 0.264)$, $P^*(C) = (0.338, 0.662)$. We obtain $P^*(F)$ by calculating $P^*(F, S)$ with $P^*(F, S) = P(F, S) * P^*(S) / P(S)$. Results are shown in Table 3.6

Table 3.6 Updated probability distribution for $P^*(F, S)$

$P^*(F, S)$	S = Summer	S = Other
F = Suitable	0.736	0.0528
F = Unsuitable	0	0.2112

By marginalization, we have $P^*(F) = (0.788, 0.212)$, which means that the probability that fishing is suitable is 0.788.

Now we apply the linear estimate propagation to this problem. First, we define a DGG for each attribute, with only a bottom node representing the domain and a top node *All*. Then we calculate the initial estimates for the bottom node in GenSpace graph with the formula $P(S,C,F,T) = P(S)P(C)P(T \mid S,C)P(T \mid S)$. The results are shown in Table 3.7.

Table 3.7 Initial and revised estimates for the bottom node

S	C	F	T	Initial Exps	Revised Exps
Summer	Victoria	Suitable	High	0.02	0.0735
Summer	Victoria	Suitable	Low	0	0
Summer	Victoria	Unsuitable	High	0	0
Summer	Victoria	Unsuitable	Low	0	0
Summer	Montreal	Suitable	High	0.18	0.6618
Summer	Montreal	Suitable	Low	0	0
Summer	Montreal	Unsuitable	High	0	0
Summer	Montreal	Unsuitable	Low	0	0
Other	Victoria	Suitable	High	0.0144	0.0529
Other	Victoria	Suitable	Low	0.0016	0
Other	Victoria	Unsuitable	High	0.0576	0.2118
Other	Victoria	Unsuitable	Low	0.0064	0
Other	Montreal	Suitable	High	0	0
Other	Montreal	Suitable	Low	0.144	0
Other	Montreal	Unsuitable	High	0	0
Other	Montreal	Unsuitable	Low	0.576	0

To perform revision, we first convert the evidence $T = High$ to the probability distribution P(T) = (1, 0). We propagate it to the bottom node and obtain the revised estimates as shown in the last column in Table 3.7. By summarization, we get $P(S)$ = (0.736, 0.264), $P(C)$ = (0.338, 0.662), and $P(F)$ = (0.788, 0.212), which are consistent with the Bayesian belief updating.

CHAPTER 4

ESTIMATE PROPAGATION IN GENSPACE SUBGRAPHS

In Chapter three, we proposed a linear GSEP method that propagates the user's new estimates to all the other nodes in a GenSpace graph. The bottom up propagation in GSEP process is similar to the aggregation process in OLAP systems. However, compared with the other summarization methods [Agarwal et al., 1996; Ross and Srivastava, 1997; Beyer and Ramakrishnan 1999], our summary mining faces the following computation challenges.

First, most other summarization methods do not involve concept hierarchies, or involve concept hierarchies that represent a totally ordered relation. Our method is based on a GenSpace graph that represents a partially ordered relation. Therefore, dynamic sorting method that helps improve the efficiency of other aggregation methods do no help with our problem.

Secondly, traditional summarization methods only aggregate data, while our method needs to aggregates data as well as to calculate interestingness measures, which requires more computation.

Thirdly, traditional summarization methods only aggregate data once, while our method needs to propagate estimates and calculate interestingness measures repeatedly.

However, there are two features of our problem that we can take advantage of. First, before the mining process, the user can specify uninteresting nodes in the data set.

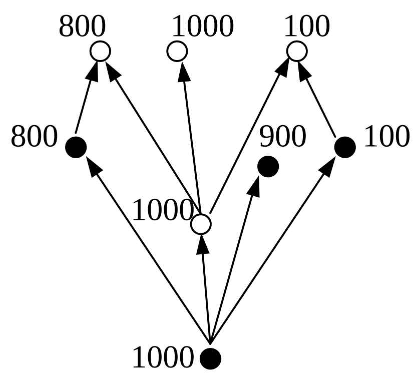
Oftentimes, the user may not be interested in lower level nodes and nodes with specific real and time values. It is these nodes that constitute the most space and time costs for the mining process. After properly pruning these nodes, we can greatly improve the mining efficiency. Second, in our problem, we only need to find top *k* interesting nodes to present to the user, therefore, we do not need to materialize all the nodes in memory at the same time. In this chapter, we propose two pruning methods for GSEP in GenSpace subgraphs.

4.1 Aggregation in GenSpace Graphs and Subgraphs

In the GSEP process, most of the time cost is caused by the bottom up propagation. The bottom up propagation is very similar to the aggregate process of data cubes, in which efficiency is a key issue. There are two kinds of popular aggregation methods for constructing a data cube: general to specific and specific to general. General to specific method excels for iceberg cube construction, where the measure satisfies the monotonic property, and thus supports Apriori-like pruning. In our case, none of the interestingness measures for summaries that we can use possesses this property; therefore, general to specific aggregation does no good in our case. The bottom up propagation takes advantage of dynamic sorting and the smallest parent node to improve efficiency. The sorting technique is useful to improve the aggregation efficiency in traditional data cube calculation when the child node is a prefix of its parent node. However, in the GenSpace graph, most of the parent-child node pairs do not satisfy the prefix relations. Therefore, sorting summaries dynamically will not improve efficiency in our case. We use the smallest parent technique in our approach. The problem becomes choosing appropriate path in GenSpace graph or subgraph to improve propagation efficiency.

An ExGen graph for an attribute can be comprehensive. For example, Hamilton et al. identified 27 nodes for the Time attribute [Randall et al., 1999]. Considering the combinational nature of the GenSpace graphs, the complete GenSpace graph may have tremendous space and time cost. However, in many occasions, the user is only interested in a subset of the nodes in a GenSpace graph and can specify some nodes as uninteresting before the propagation process starts. For example, he can specify a specific node, or a set of nodes with a specific attribute at a certain level, or a set of nodes with a certain depth in the GenSpace graph, as uninteresting. Practically, the uninteresting nodes are usually in the lower levels of the GenSpace graph, because with too many values, it will be difficult for users to grasp the insight of the table. Therefore, even if a small percentage of the nodes are pruned, the savings of space storage and propagation time may be high.

Another situation for pruning nodes occurs when one node has the same size as its smallest parent. We can see that there is no generalization effect in this situation, and thus



mark this node as uninteresting. Furthermore, if the interestingness measures we use only involve the records present in the nodes, the interestingness measure for the two nodes are the same. Generalizing this situation one more step, we can mark uninteresting a node's all descendents that have the same size with the node. In Figure 4.1, the solid ovals denote the potentially interesting nodes and the blank ovals denote the uninteresting nodes. The storage cost for the GenSpace graph is 5700 units, while the storage cost for the subgraph is 2800 units.

Figure 4.1 Uninteresting nodes

When uninteresting nodes are identified either by the user or automatically, we can prune them before propagation and select the appropriate path for propagation.

4.2 Prune Nodes While Preserving One-Step Generalization Links

We can prune all the uninteresting nodes by connecting all their parent nodes to their child nodes, but this will destroy the one-step-generalization feature of a link, and thus improve the complexity of the data structure and generalization process. Further more, pruning all uninteresting nodes may increase the time cost for the propagation. In this section, we propose a pruning method that preserves one-step generalization.

In this section, we take into account both the storage costs and time (scanning) costs when we tackle the propagation efficiency. We use the number of records (or size) in a summary to represent its storage cost. Harinarayan et al. proposed a linear time cost model for aggregating a table [Harinarayan et al., 1996]. They found that the time cost (or scanning cost) for a summarization is directly proportional to the size of the raw table. Therefore, in our case, when we propagate estimates from node *A* to node *B*, the time cost is directly proportional to the size of node *A*. We use the number of the records in *A* to represent the time cost of propagating estimates from *A* to *B*.

4.2.1 Algorithm

Example 4.1 (Figure 4.2) illustrates this problem.

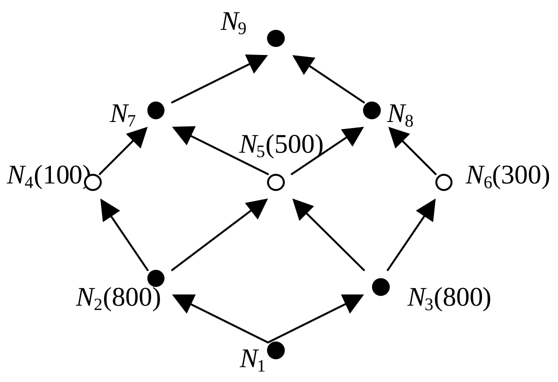


Figure 4.2 Example 4.1

Example 4.1. The blank nodes (nodes N_4, N_5, and N_6) denote the uninteresting nodes and the solid nodes denote the potentially interesting nodes. The numbers in the parentheses denotes the storage cost of the nodes. We intend to prune a subset of uninteresting nodes such that all the potentially interesting nodes can be reachable from bottom node. We can prune either N_4 and N_6 or node N_5 to satisfy this constraint. If we want to reduce storage cost, it is apparent that we prefer to prune N_5 instead of N_4 and N_6, because we can save (500 - 100 - 300) = 100 units storage. If we want to reduce time cost, we prefer to prune N_4 and N_6 and keep N_5, since we can save (800 + 800 + 100 + 300) – (800 + 500 + 500) = 200 units scanning cost.

The ***node preservation problem*** is to find a subset of the uninteresting nodes in a GenSpace graph such that every interesting node can be reachable from the bottom node and the comprehensive cost $w_sC_s + w_tC_t$ is minimum, where C_s and C_t represent the storage cost and the time cost, respectively, and w_s and w_t represent the weights.

If w_t equals to 0, the node preservation problem becomes a directed Steiner tree problem, which is NP-complete.

In this section, we propose a heuristic approach to solve this node preservation problem. We first give some definitions that will be used in our approach.

Definition 4.1 If a non-uninteresting node only has uninteresting nodes as its parents, we call it an ***endangered node***.

In Figure 4.2, nodes N_7 and N_8 are endangered nodes. If the pruning is not properly conducted, these nodes might not obtain estimates from bottom up propagations.

Definition 4.2 The parent nodes of ***endangered nodes*** are called candidate nodes.

We need to select a subset of the candidate nodes to prune and preserve the rest in the GenSpace graph, although we hide them from showing to the user.

Definition 4.3 A candidate node’s endangered children are called the candidate node’s ***cover***.

In Figure 4.2, node N_4’s cover is N_7; node N_5’ s cover is N_7 and N_8.

We use Example 4.2 to illustrate the heuristics.

Example 4.2. Figure 4.3 illustrates the definitions.

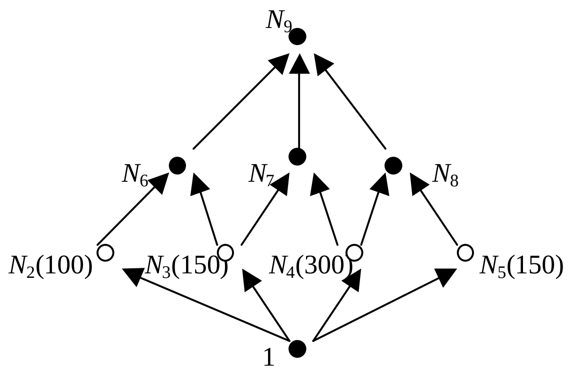


Figure 4.3 GenSpace graph for Example 4.2

Endangered node set {N_6, N_7, N_8}, Candidate node set {N_2, N_3, N_4, N_5}. $Cover_{N2}$ = {N_6}, $Cover_{N3}$ = {N_6, N_7}, $Cover_{N4}$ = {N_7, N_8}, $Cover_{N5}$ = {N_8}. We represent this in Table 4.1.

Table 4.1 Relations between the endangered nodes and the candidate nodes

Endangered / Candidate	N_6	N_7	N_8
N_2	100		
N_3	150	150	
N_4		300	300
N_5			150

In Table 4.1, each row describes a candidate node, and each column describes an endangered node. If a candidate is a parent of an endangered node, we put the storage cost in the corresponding cell. Here {N_3, N_5} is a subset that covers the entire set of endangered nodes, and its storage cost (150 + 150 = 300) is minimum. Therefore, we choose to keep N_3 and N_5 and prune nodes N_2 and N_4.

We first use a ***Linear programming*** (***LP***) method to solve this problem. The problem for LP can be formalized as follows:

Maximize or miminize the objective funtion $f(X_1, X_2, \ldots, X_n) = c_1X_1 + c_2X_2 + \ldots + c_nX_n$, such that the constraints $a_{i1}X_1 + a_{i2}X_2 + \ldots + a_{in}X_n \le b_i$, $i = 1, \ldots, m$ are satisfied, where c_i, a_{ij}, and b_i are coefficients, and the X_i are variables.

We translate the node selection problem into this format to use the LP method. In the case of *m* candidate nodes and *n* endangered nodes, we define *m* binary variables x_i corresponding to candidate nodes. If a candidate node is selected, *x* has the value of 1, otherwise it has the value of 0. We define *n* constraints corresponding to the endangered nodes. For the *j*th endangered node, the constraint is that the sum of the variables corresponing to the candidate nodes that cover the endangered node should be greater than or equal to 1, i.e., it is covered by the selected nodes. To simplify the problem, we only take into account the storage cost, i.e., set w_s to 1 and w_t to 0. Therefore, the target function to minimize is the size of the selected candidate node.

In Table 4.1, we define four binary variables $x_1, \ldots, x_4$ correponding to nodes $N_4, \ldots, N_7$. The three constraints are

$x_1 + x_3 \ge 1$,

$x_2 \ge 1$,

$x_1 + x_4 \ge 1$,

and the target function is $f(x_1, x_2, x_3\ x_4) = 10 * x_1 + 15 * x_2 + 8 * x_3 + 8 * x_4$.

Linear programming can find the optimal solutions. However the time complexity is not only related to the number of the variables and constraints, but also related to the coefficients present in the objective function and the constraints. In the worst case, the time complexity is exponential.

Here we propose heuristics to solve this problem. The intuition is that we would like to choose a smaller number of nodes with smaller sizes to preserve. Since we want to choose the nodes that have smaller storage costs and greater coverage, it is natural to create an index Storage-Coverage-Ratio (*N*) = Storage (*N*) / Coverage (*N*) for each candidate node *N*. In each step, we select a node with the highest storage coverage ratio to preserve. After the selection, we remove the endangered nodes that are covered by the selected node, i.e., we delete their corresponding columns in the table. We also delete the row in the table corresponding to the selected node. Then we recalculate the storage coverage ratio in the new table and repeat the selection process. This process continues until all the columns of the table are deleted, i.e., all the endangered nodes are covered. After obtaining the subset, we use forward adding backward elimination to eliminate the redundant nodes. Figure 4.4 presents the algorithm.

One difficulty is that this process occurs before materializing the nodes. So it is difficult to obtain the storage cost for the nodes. We use the maximum storage cost, i.e., the product of the cardinalities of the possible values for each attribute, to estimate the real cost.

Let m and n denote the numbers of the candidate and endangered nodes, respectively. The worst case occurs when only one more endangered node is covered when selecting one candidate node. If $m < n$, the number of loops is $m + (m - 1) + \dots + 1$, therefore the worst case complexity is $O((m + 1)\, m / 2) = O(m^2)$. If $m \geq n$, the number of loops is $m + (m-1) + ... + (m - n + 1) = mn - n(n-1)/2$. Therefore, the worst case complexity for this algorithm is $O(mn\text{-}n(n - 1) / 2) = O(m(m - n))$.

```
Function SelectCandidateNodesOneStep
1. Search the GenSpace graph and find all the endangered nodes.
2. Find the set of candidate nodes corresponding to the endangered
nodes.
3. Construct the coverage relation table (as in Table 1).
4. While the set of the endangered node set is not empty,
    4.1 Select a candidate node with the minimum storage coverage
    ratio. If there is a tie, select one with greater coverage. If there is a
    tie again, we randomly select one.
    4.2 Eliminate the covered endangered nodes from the endangered
    node set to eliminate the selected candidate node from the
    candidate node set.
    4.3 Recalculate the coverage and storage coverage ratio for the left
    candidate nodes.
5. Check the selected node set one by one in the reverse order of
selection using forward adding backward elimination strategy and
eliminate the redundant nodes.
```

Figure 4.4 Function SelectCandidateNodesOneStep

In Example 4.2, since node N_3 has the minimum storage coverage ratio (150 / 2 = 75), we select it and eliminate nodes N_6 and N_7 of the endangered node set. In the next step, we choose N_5, because it has minimum storage coverage ratio 150. (Initially, the storage coverage ratio (N_4) = 150, but after eliminating N_7, the ratio becomes 300). In this case, N_3 and N_5 are selected. And it is easy to see that none of them can be eliminated. Hence we keep them and prune node N_2 and N_4.

After we select a new uninteresting node to preserve, it can become an endangered node again. We have to guarantee that all the newly selected candidate nodes are safe for propagation. We adapt the GenSpace graph in Figure 4.2 to Figure 4.5 by setting nodes N_2 and N_3 as uninteresting nodes. Applying algorithm 1, we first choose N_5. Then, N_5 becomes an endangered node, because N_2 and N_3 are uninteresting nodes. We need to apply the algorithm *SelectCandidateNodesOneStep* again to select N_2. Finally, we prune nodes N_4, N_6 and N_3. Figure 4.6 presents the algorithm to select candidate nodes throughout the GenSpace graph.

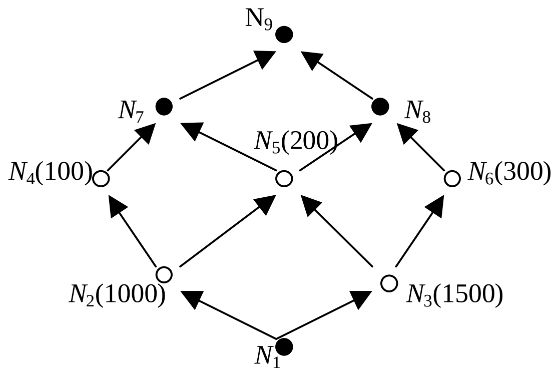


Figure 4.5 Adapted GenSpace graph of Example 4.1

```
Algorithm SelectCandidateNodes
1.Create the set of endangered nodes by scanning the GenSpace graph.
2. While the set of endangered nodes with is not empty,
    2.1 Find the nodes to preserve using Function SelectCandidateNodesOneStep.
    2.2 Find the set of endangered nodes in the selected node set.
```

Figure 4.6 Function SelectCandidateNodes

In some cases, even when there are no endangered nodes, selecting extra uninteresting nodes may reduce the time cost.

Example 4.3. In Figure 4.7, only node 2 is specified as uninteresting node. If we delete it, we have total scanning cost 2000 + 1500 * 3 + 100 = 6600. If we keep it, we have total scanning cost 2000 * 2 + 300 * 3 + 100 = 5000. This situation occurs when small sized nodes (node 2 in this example) are specified as uninteresting nodes, and the next best parents (node 3) for their children are significantly larger than the uninteresting nodes. Practically this situation is rare, because first, the users usually specify big sized nodes as uninteresting nodes.

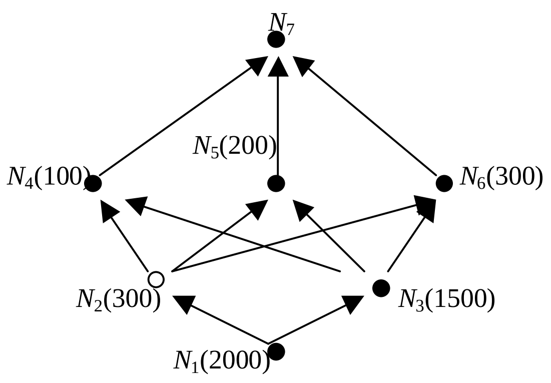


Figure 4.7 Pruning deteriorates the time efficiency

To deal with this special case, we need to select some redundant nodes that will improve efficiency after calling function SelectCandidateNodes. The idea is straightforward. We check the remaining uninteresting nodes left by function SelectCandidateNodes. In the increasing order of size, calculate the efficiency improvement. We first obtain the set of its children that have to be preserved. Then we calculate the best efficiency cost for these children into value *oldcost*. Then we calculate the efficiency cost for this node *newcost* (size of its best parent plus the best efficiency cost for its children considering the presence of this node). If *oldcost* – *newcost* is greater than a nonnegative threshold, we keep this node, otherwise we prune it. This process continues until all the nodes are checked. In Figure 4.7, the *oldcost* is 1500 * 3 = 4500. The *newcost* is 2000 +300 * 3 = 2900. The difference is 1600. If we set threshold to be 1000. We keep this node. The algorithm is presented in Figure 4.8

```
Algorithm One-Step-Generalization-Pruning
1. Call SelectCandidateNodes to select nodes to preserve.
2. Find the set S of the left nodes that at least have one preserved child.
3. While S is not empty {
3.1     Find the smallest node of S.
3.2     Find the improvement value I of this node.
3.3     If I is greater than a given threshold, then keep it, otherwise, prune it.
3.4     Find the set S of the left nodes that at least have one preserved child.
}
4. Prune all the left uninteresting nodes.
```

Figure 4.8 Algorithm One-Step-Generalization-Pruning

4.2.2 Experimental Results

To measure the efficiency of the propagation in subgraphs created by the heuristic method proposed in Section 4.2.1, we first experimented on synthetic data sets, and then on the Saskatchewan weather data set and the University of Regina Student data set.

As we mentioned before, since the propagation time is directly proportional to the size of the records scanned in GenSpace graphs [Harinarayan et al., 1996], we report propagation time in thousands of records scanned in our experiments. The advantage is that this measure is independent to the detailed implementation and computers where we run the program.

We generated a set of tables with sizes ranging from 40K to 200K. All tables have four attributes, a_1, a_2, a_3, and a_4. The possible values for these attributes are integers. All the values in the table are generated randomly. For simplicity, we give all attributes identical ExGen graphs, as shown in Figure 4.9. We did two series of experiments to test the effect of cardinality and depth below which the nodes are marked as uninteresting.

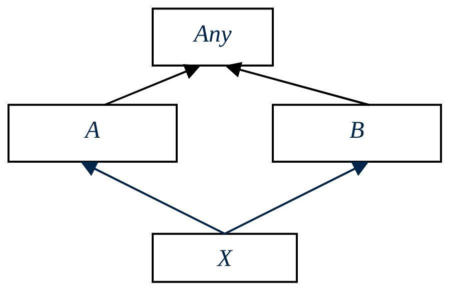


Figure 4.9 DGG for synthetic data sets

Scalability. In this series, we want to see the storage and time savings of propagation paths obtained by the proposed method to the paths consisting of only potentially interesting nodes and the paths consisting of all nodes as a function of the sizes of data sets. We set the number of sections in nodes *A*, *B*, and *X* to 5, 30, and 50 respectively and varied the size of data sets from 40K to 200K with an increment of 40K. We marked the nodes under level four in the GenSpace graph as uninteresting, which results in 162 uninteresting nodes and 94 potentially interesting nodes. Figures 4.10 and 4.11 show the time cost and storage cost for bottom nodes with different sizes. Figure 10 shows that the time cost of propagation in the GenSpace subgraph obtained by our heuristic algorithm is significantly less than those in the entire GenSpace graph and in the subgraph obtained by pruning all uninteresting nodes. Figure 4.11 shows that the storage cost of the GenSpace subgraph is significantly less than that of the entire GenSpace graph.

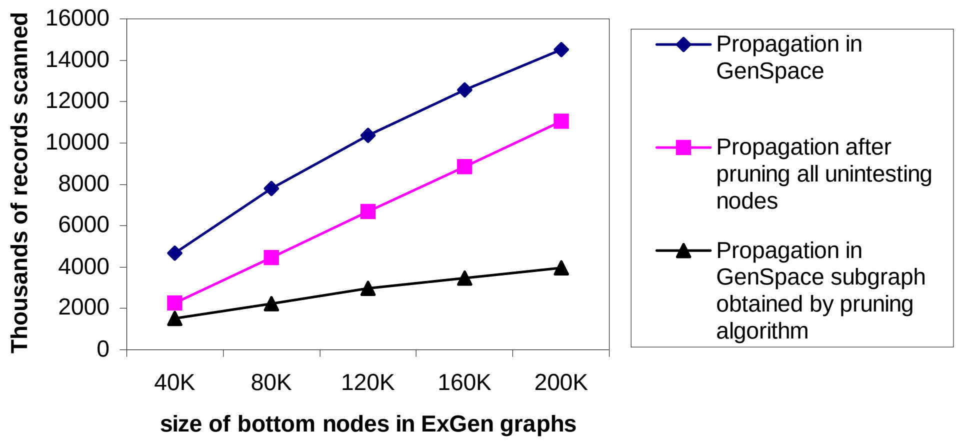


Figure 4.10 Time cost for bottom nodes with different sizes

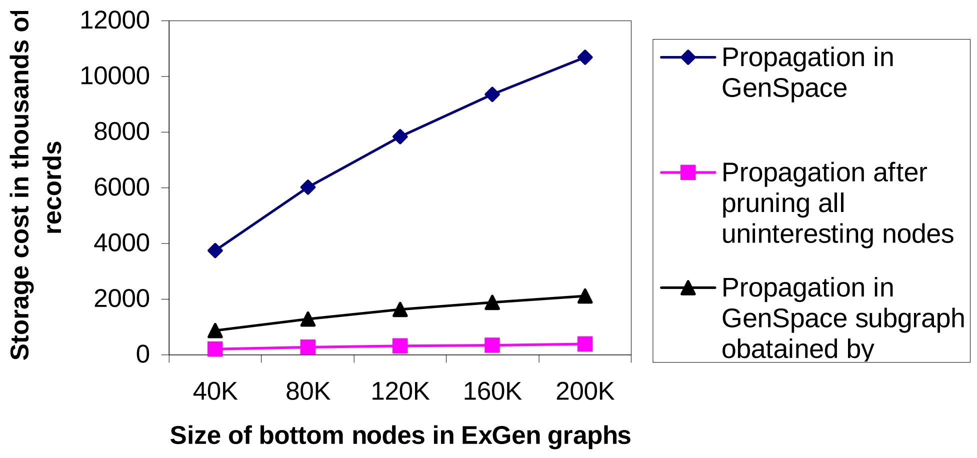


Figure 4.11 Storage cost for bottom nodes with different sizes

Depth of uninteresting nodes. In this series, we mark nodes at varoius depths as uninteresting. We set the size of *A*, *B*, and *X* to 8, 8, and 50. The size of the data set is 200K. Figures 4.12 and 4.13 compare the time and storage costs, respectively. As we expected, when we mark uninteresting nodes below very low levels, the time savings are limited, because no nodes or only a few nodes can improve the propagation time; while when we set uninteresting nodes from the middle levels, the time savings are significant.

We also observed two interesting phenomena. First, when we mark the nodes under level 1 as uninteresting, the subgraph obtained with the pruning algorithm has more propagation costs than that obtained from pruning all uninteresting nodes. This is because the nodes at level 1 are very large, or even as large as the bottom node. Incorporating some nodes in that level will not reduce the propagation costs to the nodes at level 2, while it introduces the extra propagation costs for itself. Second, we observed that when we prune all uninteresting nodes, the propagation costs increase when we increase the levels of the uninteresting node for the first two levels. This is because the number of the nodes in the lower levels increases with the level number. Pruning all the nodes in the lower levels results in propagation from the bottom node to more nodes, which is the bottleneck of the propagations.

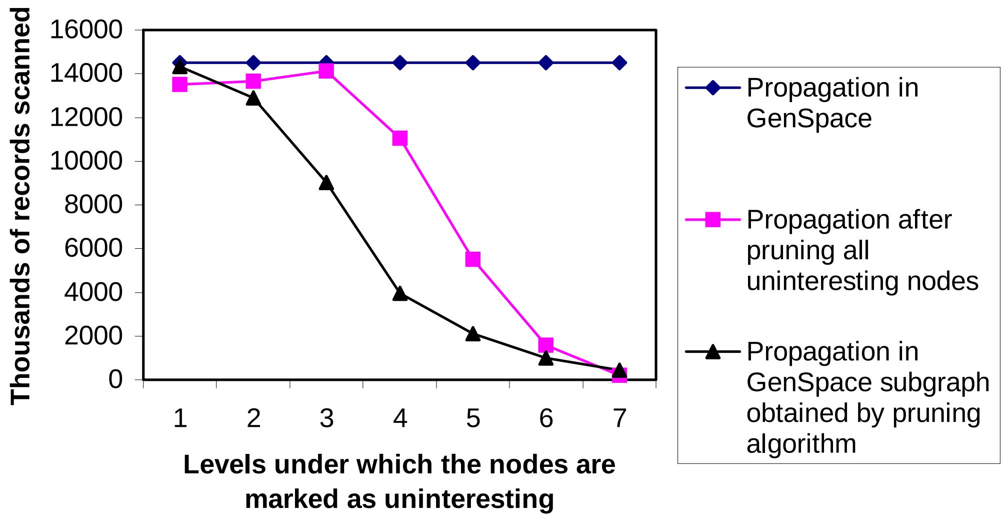


Figure 4.12 Propagation time with different levels of uninteresting nodes

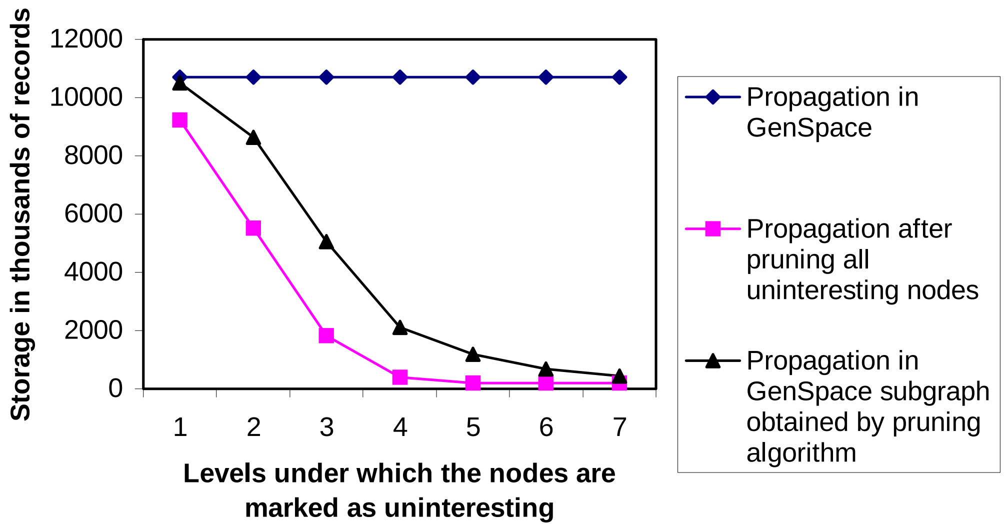


Figure 4.13 Storage cost with different levels of uninteresting nodes

For the Saskatchewan weather data set, we assume two scenarios. First, we assume that all the nodes with depth less than or equal to four in GenSpace graph are not interesting. In this case, 165 out of 560 nodes are uninteresting. In the second scenario, we assume that all the nodes with specific date and specific temperature values are not interesting. In this case, there are 200 uninteresting nodes. Figure 4.14 shows that the storage space (in the number of records) for uninteresting nodes and the storage for preserved nodes using heuristics and linear programming in the first scenario, with the bottom nodes ranging from 40K and 200K. Figure 4.15 shows the scanning costs for the GenSpace graph with and without pruning. Figure 4.16 and 4.17 show the corresponding trends for scenario 2. We can see that the storage and scanning costs for both cases have linear trends. We compared the results for linear programming and the proposed heuristics. For the linear programming algorithm, we use the commercial tool Premium Solver Platform Version 5.0 with Large-Scale LP Solver Engine (Frontline Systems, Inc. Website: http://www.solver.com/). The results for linear programming and heuristic selection are very close. For case 1, the storage cost obtained from linear programming is consistently less than that obtained from heuristics, but the difference is trivial. The scanning cost obtained from linear programming and the heuristics are intertwined. For case 2, the results from two methods are identical. Although the results of linear programming and heuristics are very close, their running time differs significantly. For the heuristics selection, the running time is unperceivable regardless of the size of the bottom node, while in linear programming, the running time changes dramatically according to coefficients. In scenario one, when the size of bottom node is 40K, the running time is a couple of seconds; when bottom node increases to 80km, the running time increases to 10 minutes; when the bottom node increases to 160k, the running time is 8 hours; when the bottom node increases to 200k, the program runs out of memory.

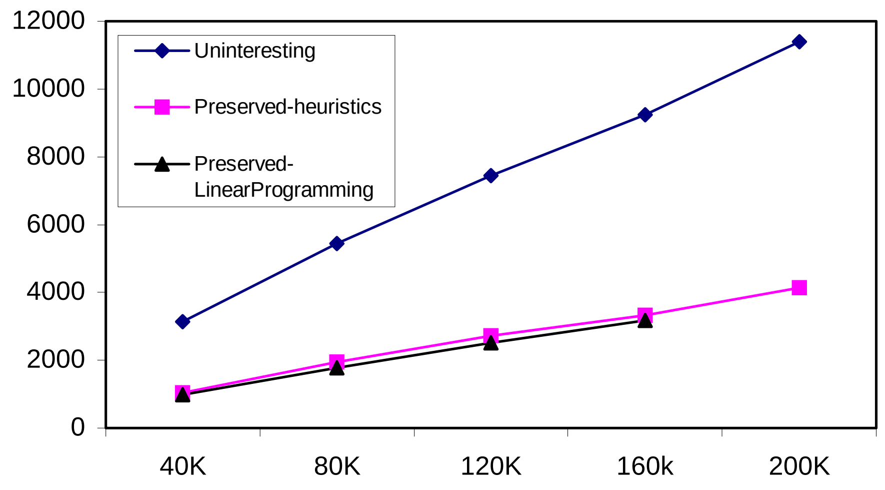


Figure 4.14 Storage of uninteresting nodes versus storage of preserved nodes for scenario 1 for Saskatchewan weather data set

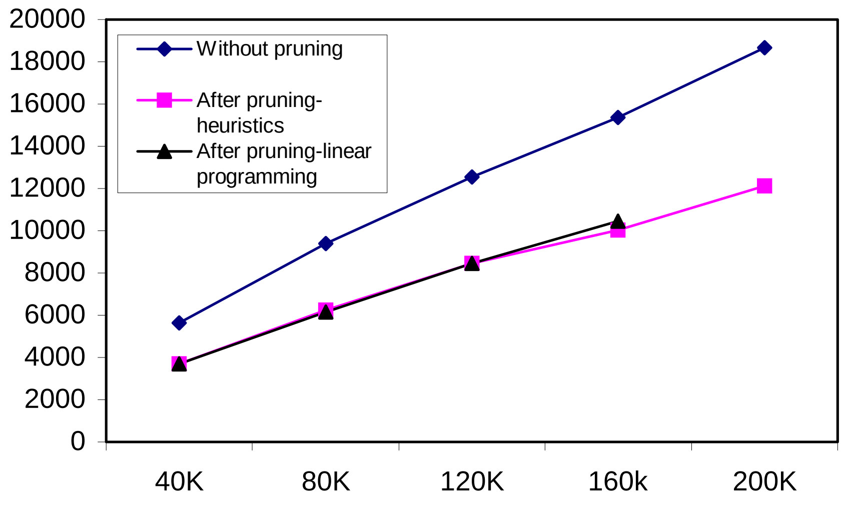


Figure 4.15 Scanning cost of unpruned graph versus pruned graph for scenario 1 for Saskatchewan weather data set

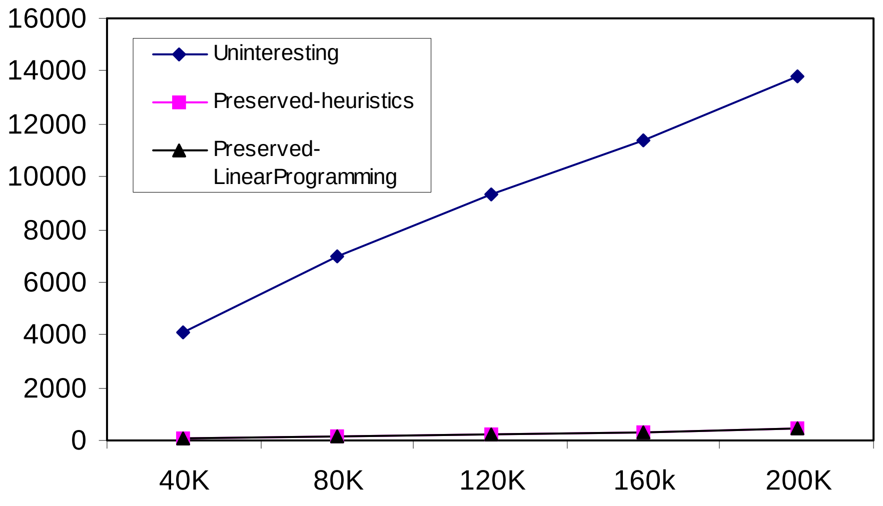


Figure 4.16 Storage of uninteresting nodes versus storage of preserved nodes for scenario 2 for Saskatchewan weather data set

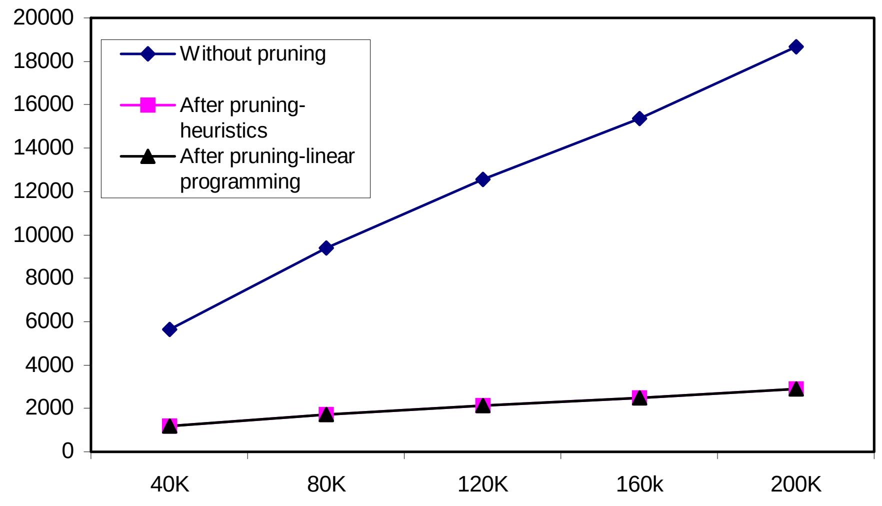


Figure 4.17 Scanning cost of unpruned graph versus pruned Graph for scenario 2 for Saskatchewan weather data set

For the student data set, we assume two scenarios. In the first scenario, we assume that all the nodes with depth less than or equal to four in the GenSpace graph are not interesting. In second scenario, we assume that all the nodes involving any specific values are not interesting. Figure 4.18 shows that the storage space for uninteresting nodes and the storage for preserved nodes using the heuristics and linear programming in the first scenario, with the bottom nodes ranging from 10K and 60K. Figure 4.19 shows the time costs for the GenSpace graph with and without pruning. Figure 4.20 and 4.21 show the corresponding trends for scenario 2.

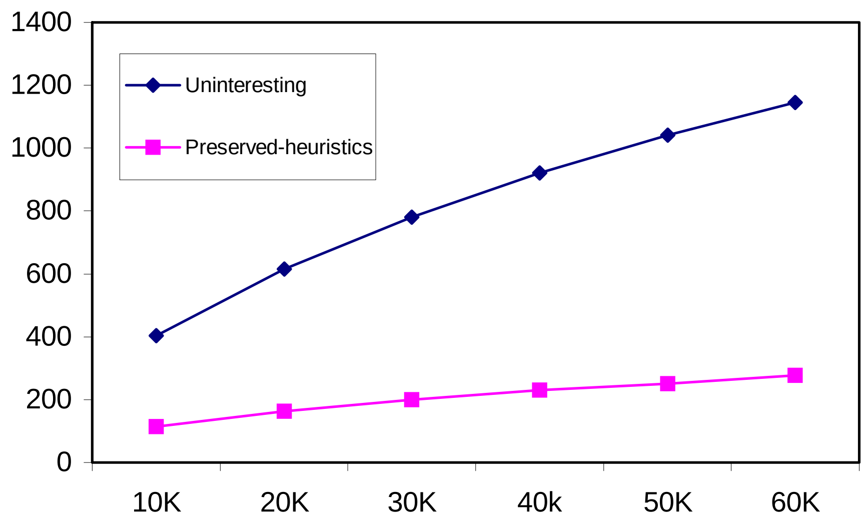


Figure 4.18 Storage of uninteresting nodes versus storage of preserved nodes for scenario 1 for student data set

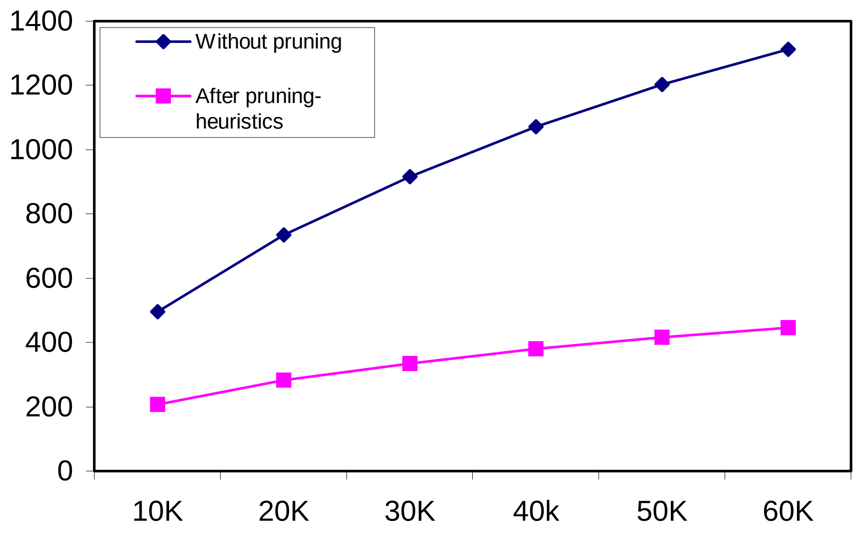


Figure 4.19 Scanning Cost of unpruned graph versus pruned graph for scenario 1 for student data set

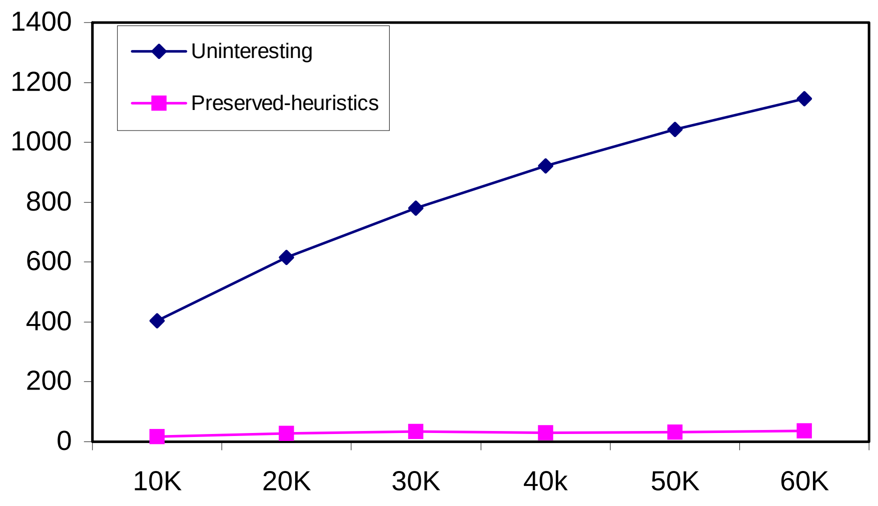


Figure 4.20 Storage of uninteresting nodes versus storage of preserved nodes for scenario 2 for student data set

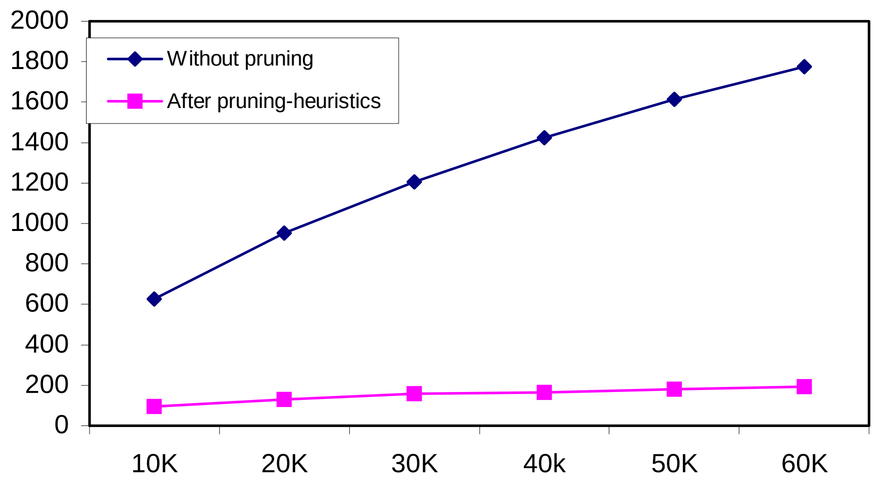


Figure 4.21 Scanning cost of unpruned graph versus pruned graph for scenario 2 for student data set

4.3 Prune Nodes and Incorporate Multi-Step Generalization Links

In this section, we propose a pruning method that incorporates multi-step generalization links. Compared with the pruning method involving one-step generalization links, the subgraph obtained from this pruning method requires less memory, but with more complex data structures and time for generalizing attributes.

4.3.1 Algorithm

Example 4.4 illustrates the problem.

Example 4.4. In Figure 4.22, the solid ovals denote interesting nodes and the blank ovals denote uninteresting nodes. If we prune node N_4, the propagation cost from node N_5 to N_1, N_2, and N_3 is size(N_5) * 3 = 3000. If we preserve N_4 as a hidden node, the propagation cost is size(N_5) + size(N_4) * 3 = 1600. Keeping hidden nodes reduces the propagation cost by nearly 50%.

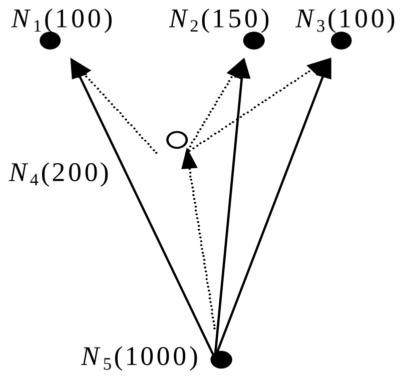


Figure 4.22 Efficiency improvement due to keeping uninteresting nodes

Sarawagi et al. encountered a similar problem when they calculated a collection of group-bys [Sarawagi et al., 1996]. They converted the problem into a variant of the directed Steiner tree problem, which is an NP-complete problem. They used a heuristic method proposed in [Smith and Liebman, 1980] to solve their problem. In order to use the Steiner tree model, they need to create additional links in the graph to connect all pairs of nodes that have a generalization relation, which causes extra storage space. In the extreme case, for example, for a totally ordered relation with 2000 nodes, the number of the original links in GenSpace is 1999, and the number of the expanded links is 1,999,000.

Furthermore, when we materialize all interesting nodes in memory to improve propagation efficiency, the storage space is limited for the uninteresting nodes. The Steiner Tree problem does not impose constraints on the size of the nodes.

Here, we consider our problem as a constrained directed Steiner tree problem, where there is a space limit for the uninteresting nodes. We propose a greedy method to select hidden nodes and find an efficient propagation path for subgraphs.

Definition 4.4. Given a GenSpace graph $G = <P, Arc, E>$ and a set of nodes $N \subseteq P$ such that the bottom node $X \in N$, the ***optimal tree*** OT of N in G is a tree that consists of nodes in N. The root of the tree is X. For every non-bottom node $n \in N$, its parent is its smallest ancestor in G that belongs to N.

In Figure 4.23(a) is a GenSpace with uninteresting nodes. Figure 4.23(b) is the optimal tree for the potentially interesting nodes.

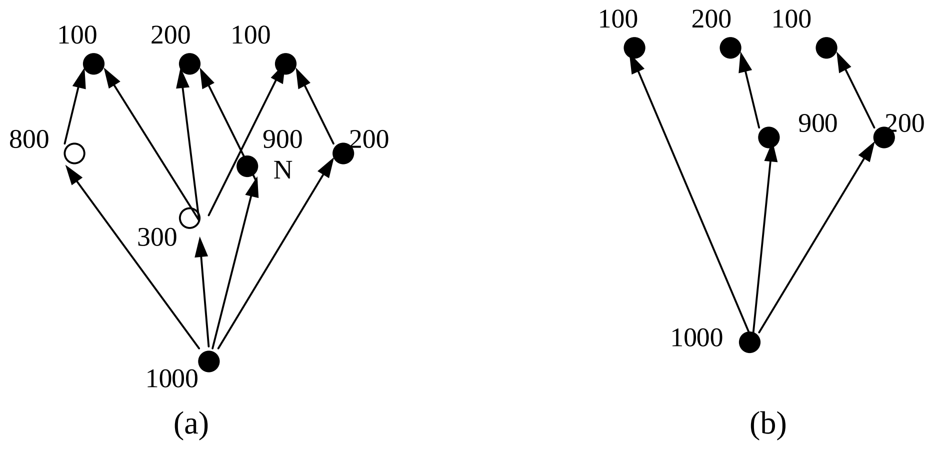


Figure 4.23 Optimal path for potentially interesting nodes

The optimal tree for a set of nodes has the most efficient propagation paths for the subgraph consisting of these nodes, because every node obtains estimates from its smallest possible ancestor in the subgraph.

Our method for choosing hidden nodes and determining the propagation paths is given in Figure 4.24. We first create the optimal tree consisting of only the interesting nodes, then check the uninteresting nodes, and select the one that results in the greatest reduction in the scanning cost. Then we modify the optimal tree to incorporate the selected nodes. This process continues until the memory limit is reached, no improvement can be obtained, or no candidate uninteresting nodes are left. Selecting an uninteresting node will affect propagation efficiency in two ways. First it will cause an extra propagation cost for propagating estimates to this node; secondly, it may reduce the cost for its interesting descendents. Function Improvement calculates the efficiency improvement for an uninteresting node *u*. If it returns a positive value, it will reduce the scanning cost.

Let *m* denote the number of uninteresting nodes and *n* denote the number of all nodes in a GenSpace graph. In the worst case, all the uninteresting nodes are selected to be preserved. In the first round, we calculate the improvement of *m* nodes each with complexity *n*, thus the total complexity is *mn*. In the second round, we calculate the improvement of (*m*-1) nodes, each with complexity *n*. In the *k*th round, the complexity is (*m*-*k*+1)*n*, and the worst case complexity is calculated as $\sum_{k=1}^{m} n(m-k+1) = \frac{nm(m-1)}{2}$. Therefore, the complexity of this algorithm is O(nm^2).

```
SelectHiddenNodes(GenSpace graph G, Interesting node set I,
Uninteresting node set U) {
1. Create initial optimal tree OT based on I;
2. U' = ∅;
3. While there is enough memory and U is not empty do {
4.      For each u ∈U, do
5.           $Imp_u$ = Improvement(u, I, G, U', OT);
6.      Select $u_{max}$ as the one with maximum improvement $Imp_{max}$;
7.      If $Imp_{max} \le 0$
8.           break;
9.      U' = U' ∪ {$u_{max}$}; U = U − {$u_{max}$};
10.     For each node x ∈ I∪U' do {
11.          Find its best ancestor $u_{an}$ (with smallest size) in I∪U';
12.          Set $u_{an}$ as x's parent;
13.     }
14.     OT = newly generated optimal tree;
15. }
16. Return selected node set U' and optimal tree OT for nodes I∪U';
17. }

Improvement(u, I, G, U', OT) {
18. Improvement = 0;
19. Find u's best ancestor $u_{an}$ ∈ I∪U' in G;
20. Find u's descendent set D ⊆ I∪U' in G;
21. For each d in D, do {
22.     Find d's parent p in current OT;
23.     If size(p) > size (u)
24.          Improvement += size(p) – size(u);
25. }
26. Improvement –= size($u_{an}$);
27. Return Improvement;
28. }
```

Figure 4.24 Algorithm for selecting hidden nodes and creating optimal tree

Our method is similar to the method proposed in [Harinarayan et al., 1996] in that they are all greedy algorithms for selecting nodes for materialization. The difference is that the method in [Harinarayan et al., 1996] aims to select a subset of nodes to materialize in order to facilitate the calculation of a query on an arbitrary node. In other words, it intends to improve the average calculation time for a node. Our method aims to find an optimal path in the GenSpace graph such that the calculation of a subset of nodes is efficient. The improvement function of their method takes non-negative values, which means that given enough memory, the algorithm always select a node to materialize until all nodes are selected. In our algorithm, the improvement function can take negative values, since incorporating a new node will incorporate the scanning cost for materializing this node.

4.3.2 Experimental Results

To measure the efficiency of the propagation in subgraphs created by the heuristic method proposed in Section 4.3.1, we first experimented with synthetic data sets, and then on the Saskatchewan weather data set and the University of Regina student data set.

We used the same synthetic data sets and DGGs as in section 4.2.2.

Scalability. In this series, we want to see the time savings of propagation paths obtained by the proposed method to the paths consisting of only potentially interesting nodes as a function of the sizes of data sets. We set the number of sections in nodes *A*, *B*, and *X* to 5, 30, and 50 respectively and varied the size of data sets from 40K to 200K with an increment of 40K. We marked the nodes under level four as uninteresting, which results in 162 uninteresting nodes and 94 potentially interesting nodes. Figure 4.25 shows the percentage time savings when we select some uninteresting nodes to facilitate propagation. The savings increase when the size of the bottom node increases. The possible reason is that when the size of the bottom nodes increases, the generalization ratio in the GenSpace increases.

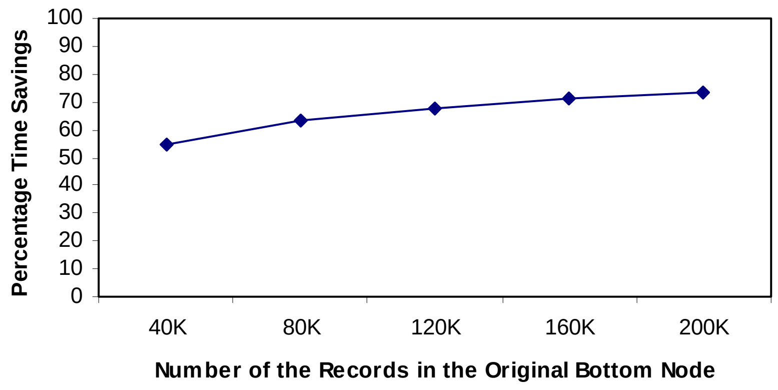


Figure 4.25 Percentage savings for data sets with different sizes

We also experimented with scalability from another perspective by changing the size of the bottom nodes in ExGen graphs as shown in Figure 4.9. We set the size of *A* and *B* to 5 and varied the size of the bottom nodes for all ExGen graphs from 10 to 80. We marked nodes under level 4 as uninteresting. The time cost and storage cost are shown in Figures 4.26 and 4.27, respectively. Figure 4.26 shows that the propagation cost in the GenSpace subgraph obtained by our approach is consistently and significantly lower than that in both the GenSpace graph and subgraph consisting of only interesting nodes. We also observed that the propagation costs of the subgraph obtained by pruning all uninteresting nodes go flat when we increase the sizes of the bottom nodes of the ExGen graphs. This is because when we increase the sizes of the bottom nodes in the ExGen graphs, the nodes in the lower levels in the GenSpace graph usually increases accordingly and the nodes in the upper levels are usually small and tend to have constant sizes. If we prune the nodes in the lower levels, the propagation costs for the upper levels will not change dramatically. The results also show that when the sizes of the bottom nodes are small (less than 55 according to the results), pruning all the uninteresting nodes will deteriorate the propagation costs. This is because when the sizes of the bottom nodes of ExGen graphs are small, the nodes in the lower levels are much smaller than the bottom node. When the sizes of the bottom nodes of ExGen graphs are large, the nodes in the lower levels are of nearly the same size as the bottom node. If we prune the nodes that are much smaller than the bottom node, which can facilitate the propagation, the propagation costs deteriorate. Figure 4.27 shows that the storage of the GenSpace subgraph does not significantly increase compared to the storage of entire GenSpace graph.

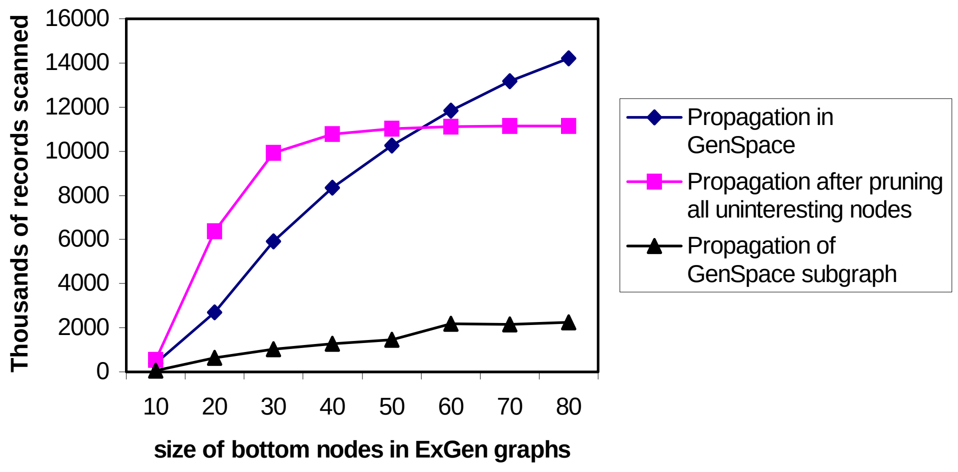


Figure 4.26 Time cost for bottom nodes with different sizes

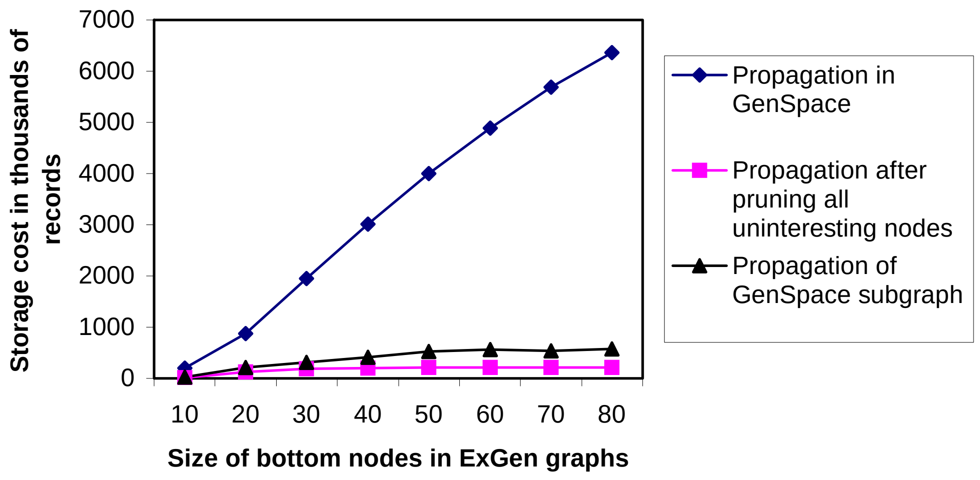


Figure 4.27 Storage cost for bottom nodes with different sizes

Depth of uninteresting nodes. In this series, we mark nodes at various depths as uninteresting. We set the size of *A*, *B*, and *X* to 8, 8, and 50. The size of the data set is 200K. Figures 4.28 and 4.29 compare the time and storage costs, respectively. Figure 4.30 shows the percentage of time saved. As we expected, when we mark uninteresting nodes below very low levels, the time savings are limited, because no nodes or only a few nodes can improve the propagation time; while when we set uninteresting nodes from the middle levels, the time savings are significant.

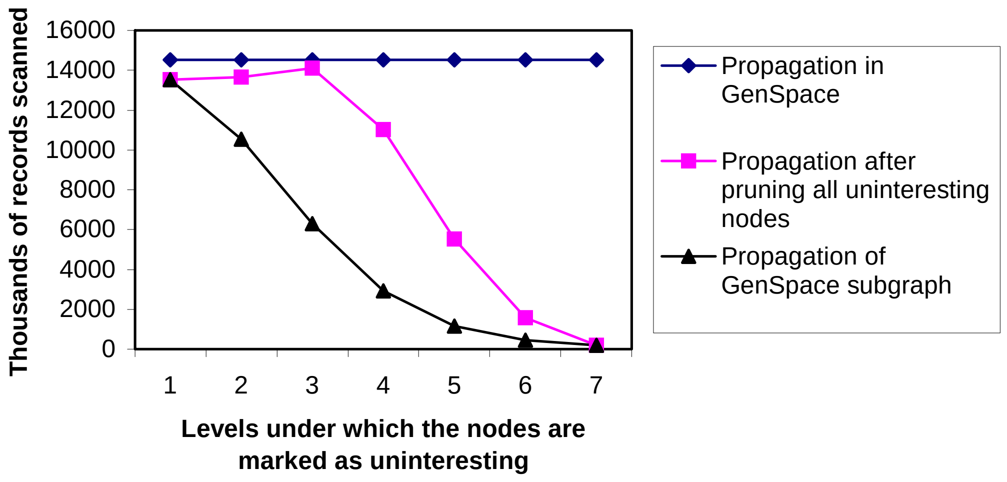


Figure 4.28 Propagation time with different levels of uninteresting nodes

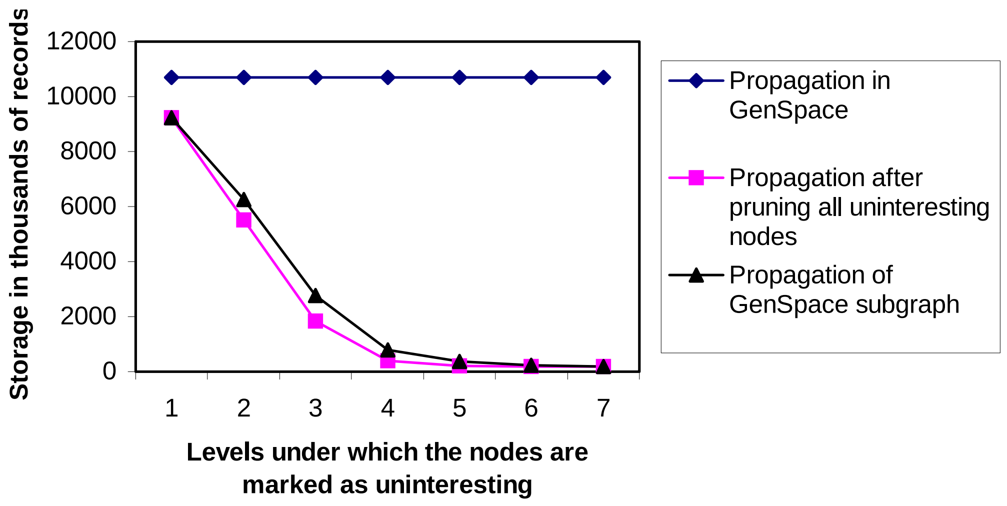


Figure 4.29 Storage with different levels of uninteresting nodes

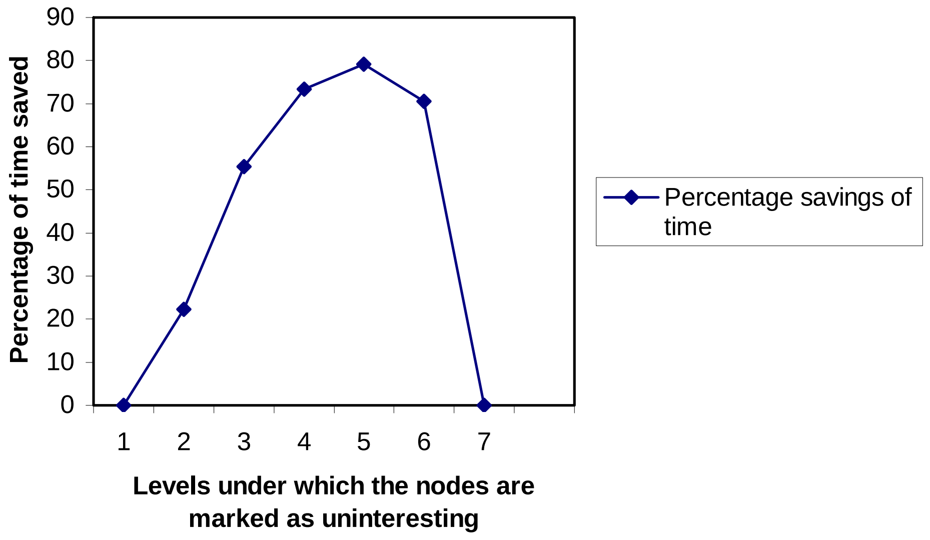
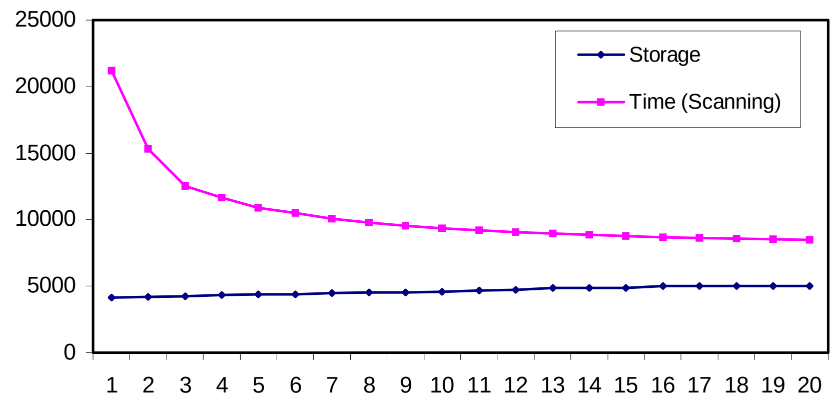


Figure 4.30 Time savings for different levels of uninteresting nodes

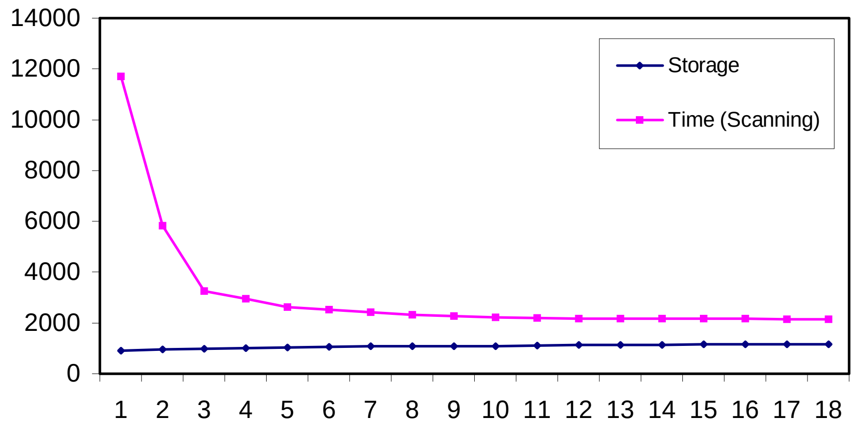
For the Saskatchewan weather data set, we experimented with three cases: (1) mark all nodes in the lowest 5 levels (out of 19 levels) as uninteresting, (2) mark all nodes in the lowest 5 levels or with specific date values or specific temperature values as uninteresting, (3) mark all nodes with specific values for any attribute as uninteresting. The results are shown in Table 4.2. The *Storage* column lists the storage cost in thousands of records. The *Scanning* column lists the number of the records scanned during propagation, in thousands of records, which gives an estimate of the propagation cost. The *Time* column lists the propagation cost in seconds. In case 1, after selecting hidden nodes, the storage increased by 25%, while the scanning cost decreased by 60%. In case 2, the storage cost increased by 26%, and the scanning cost decreased by 82%. In case 3, the storage cost increased by 8% and the scanning cost decreased by 42%. In all three cases, the scanning cost decreased significantly while the storage cost increased by a smaller percentage. Figure 4.31 illustrates the storage and scanning cost after selecting varying numbers of nodes for cases 1 and 2. The X-axis denotes the number of the nodes currently selected, and the Y-axis denotes the storage and scanning cost, in thousands of records. In both cases, the first few nodes contribute the most to the reduction of the scanning cost. Therefore, we suggest that the node selection process be halted when the improvement is below a given threshold. These experimental results indicate that with the propagation path selected by our approach, the propagation time is significantly reduced, while the increase of storage space remains acceptable.

Table 4.2 Efficiency comparison of propagation in GenSpace subgraph in three cases for Saskatchewan weather data set

		Storage (K)	Scanning (K)	Time (Sec)
Case 1	Prune all	4128	21173	2149
	Heuristic	5156	8391	1056
Case 2	Prune all	912	11703	983
	Heuristic	1152	2155	152
Case 3	Prune all	467	1692	158
	Heuristic	507	965	69



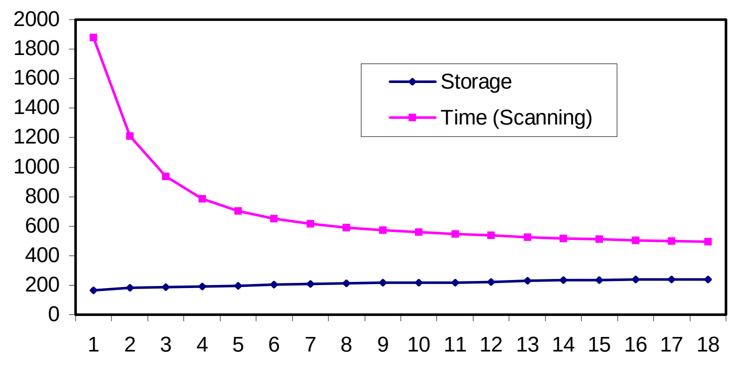
(a) Case 1



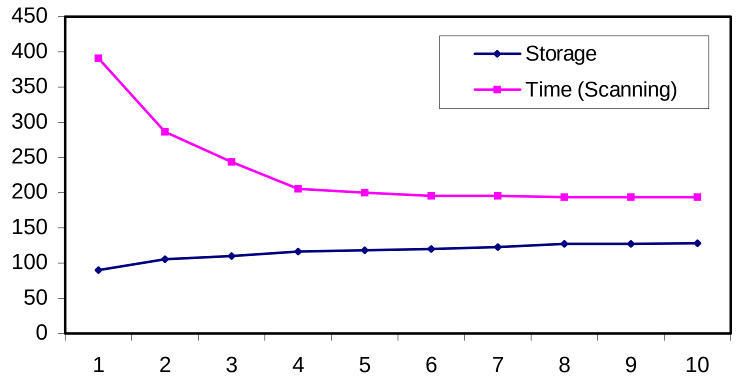
(b) Case 2

Figure 4.31 Storage and time costs when selecting hidden nodes for Saskatchewan weather data set

For the student data set, we experimented with two cases: (1) mark all nodes in the lowest 4 levels as uninteresting, and (2) mark all the nodes with attributes at specific levels as uninteresting. Figure 4.32 illustrates the storage and scanning cost after selecting varying numbers of nodes for cases 1 and 2.



(a) Case 1



(b) Case 2

Figure 4.32 Storage and time costs when selecting hidden nodes for student data set

4.4 Discussion

In most cases, subgraphs for multi-step generalization have less storage cost and scanning cost than one-step generalization subgraphs. However, we need to store additional mapping information and procedures to deal with multi-step generalization. In the previous discussion, we use the number of records scanned to describe the time cost for propagation. Now we separate the time cost into generalization cost and aggregation cost for analysis.

Generalization cost refers to the time cost of generalizing a more specific record to a more general record. Aggregation cost refers to the time cost of aggregating estimates in the more general node. Although multi-step generalization may scan fewer records, for each record, it has more generalization cost. Therefore, if the subgraph consisting of the potentially interesting nodes is already optimal or close to optimal in terms of propagation without incorporating uninteresting nodes, we intend to choose the one-step generalization method. Otherwise we choose the multi-step generalization method. Since when no or a few additional uninteresting nodes are selected, the propagation cost in the number of the scanned records are similar for both generalization methods, therefore, one step generalization is more efficient. Otherwise, the multi-step generalization method always scans fewer records than the one-step method, and is, therefore more efficient.

In our implementation of the GSEP, we tried the ***updating*** and ***generating*** methods. In the updating method, we materialize all the potentially interesting nodes in memory. When new estimates come, we update the old estimates. In the generating method, we only keep the currently unexpanded nodes and the top *k* nodes in memory and release all other nodes when we propagate the estimates to them and calculate the measure values for them. Experimental results show that when the bottom node is small, the updating method is slightly more efficient than the generating method. However, when the size of the bottom node increases, the efficiency of the updating methods deteriorates quickly. In our implementation, we adopted the generating method.

CHAPTER 5

CONSTRUCTING VIRTUAL BOTTOM NODES WITH GRANULAR COMPUTING

Bottom nodes are always the bottleneck for the GSSM process. To tackle this problem, in this chapter, we use virtual bottom nodes as substitutes for real bottom nodes. The virtual bottom node can be significantly smaller than its corresponding real bottom node. The mining results with virtual bottom nodes are identical to those obtained with real bottom nodes.

5.1 Virtual Bottom Nodes

Since an ExGen (or DGG) graph can be used to represent the ontology for a domain, it can be fairly complex. For example, in [Randall et al., 1999], a calendar DGG with 27 nodes is proposed. However, in a specific application, perhaps only a small subset of the nodes is interesting to a user. If the user can mark uninteresting nodes in ExGen graphs for individual attributes, the ExGen graphs can be pruned before the GenSpace graph is created, and the resulting GenSpace will be smaller than it would have been. Usually the bottom node is not interesting to the user, since it represents the basic granularity of the domain and has large cardinality. However, if the bottom node of an ExGen graph is marked as uninteresting, pruning the bottom node will prevent propagation from being performed consistently, because GSEP first propagates the estimates to the single bottom node before propagating them to the other nodes. We need to use the bottom node as the basis for this propagation. Preserving the structure of a

GenSpace graph with a single bottom node allows us to preserve the consistency of propagation. On the other hand, bottom nodes are usually very big and consequently form the bottleneck for storage and propagation. Reducing its size is desirable for storage and propagations in GSEP. In this section, we introduce a substitute bottom node, called a ***virtual bottom node*** (***VBN***), which can have a smaller size than its corresponding real bottom node, but still has a generalization relation with all other potentially interesting nodes in the ExGen graph. Since the propagation process may be repeated many times, creating virtual bottom nodes before the propagation process is worthwhile.

Virtual bottom node construction can be considered as a specific form of granular computing. ***Granular computing*** refers to the theories, methodologies, techniques, and tools that make use of granules, i.e., groups, classes, or clusters of a universe, in the process of problem solving [Yao, 2000; Bargiela and Pedrycz, 2002; Skowron and Stepaniuk, 2001]. A ***granule*** is composed of a set of elements drawn together by indistinguishablitiy, similarity, or proximity relations. Two major issues regarding granular computing are the construction of granules and computation with granules.

With regard to using a virtual bottom node of a GenSpace graph, the construction of the virtual bottom node can be considered to be construction of a granule, and the propagation in the GenSpace with a virtual bottom node can be considered as computing with granules. The criteria for constructing a virtual bottom node is that the output should not be changed when we replace the real bottom node with the virtual one, and the space required should be reduced. Here we look for a granulation level that is as rough as possible, but contains all the information needed for propagation.

Definition 5.1 In an ExGen graph G, let the set of interesting nodes be I. A node B is a ***base interesting node*** iff $B \in I$ and there does not exist an interesting node $A \in I$ such that $A \preceq B$.

Definition 5.2 The ***virtual bottom node*** (***VBN***) V of ExGen graph G is the combination of the partitions of base interesting nodes B_i, such that V satisfies $X \preceq V$ and $V \preceq B_i$, and there does not exist a partition V' such that $X \preceq V'$, $V' \preceq B_i$, and $V \preceq V'$.

From Definition 5.2, we can see that the VBN V is a rougher partition of the bottom node X since we have $X \preceq V$. It is the roughest possible partition that has generalization relation with every base interesting node B_i since we have $V \preceq B_i$ and there does not exist a partition V' such that $V' \preceq B_i$, and $V \preceq V'$. If there is only one base interesting node B, the VBN V is identical to B. In the worst case, $V = X$.

We call an ExGen graph with a VBN a ***virtual ExGen graph*** and a GenSpace graph with a VBN a ***virtual GenSpace graph***.

Here we seek indiscernibility relations based on different levels of generalizations for our granule construction. Two objects are indiscernible with respect to condition attributes B if they have the same values for all the attributes belonging to B [Pawlak, 1992]. Yao et al. proposed a granulation based on indiscernible relations on multiple attributes [Yao, 2000], while our indiscernible relations are based on one attribute, but multiple generalizations of that attribute.

Figure 5.1 gives our algorithm for constructing a VBN for an ExGen graph.

There are four input parameters for this algorithm, the attribute we work on, its corresponding ExGen graph, the set of base interesting nodes, and the original data table. The first two lines of the algorithm deal with the situation where there is only one base interesting node. In this case, we return the base interesting node as the VBN. If there are two or more base interesting nodes in the ExGen graph, we initially set the VBN as an empty set. Then for each real value in the real bottom node, we map the value into all the base interesting nodes, and concatenate the results to get a virtual value. Then we uniquely insert the value into the VBN.

The worst case complexity for this algorithm is $O(n(m + n))$, where n is the cardinality of the bottom node *BT* and m is the cardinality of *BI*.

```
Algorithm ConstructVB

Input: attribute a, its corresponding ExGen graph EG, the set of base
interesting nodes BI, bottom node table BT.
Output: virtual bottom node VB.
1. If there is only one node b in BI,
2.    Return VB = b;
3. VB = {};
4. For each value bvalue in BT do
5.    VirtualValue = “”;
6.    For each base interesting node b ∈ BI do
      // Find the generalized value of bvalue in b.
7.    GenValue = FindGenValue (bvalue, b);
      // Concatenate the generalized value to the current virtual value.
8.    VirtualValue = Concatenate(VirtualValue, GenValue);
9.    If VirtualValue is not in VB
10.       Insert VirtualValue into VB;
```

Figure 5.1 Algorithm for constructing a VBN

After we obtain the VBN, we connect it to all the base interesting nodes as a parent. We then remove the real bottom node and all the uninteresting nodes that have base interesting nodes as decedents. We preserve the other uninteresting nodes in the ExGen graph to construct the GenSpace graph. After obtaining GenSpace graph, we mark the nodes that are composed of any uninteresting nodes and VBNs in ExGen graphs as uninteresting automatically, and use the algorithms proposed in Chapter 4 to prune nodes in the GenSpace graph.

In the linear GSEP process, if old estimates are non-zero, the results for the virtual GenSpace graph are identical to that for the original GenSpace graph. However, if there are zeros for old estimates, we need to adjust the propagation formula to make the propagation results identical in both the virtual and original GenSpace graphs. In this case, we record the size of each sections in the VBN, i.e., count the number of elements in the real bottom node belonging to each section in the VBN, denoted as *size*(v) = |*Spec*(v, X)|, where v is a section in VBN, and X denotes the real bottom node. If the old estimate for the *k*th section in the *i*th node equals zero, i.e., $E\_old(S_{ik}) = 0$, we propagate the new estimate $E(S_{ik}) = E_{ik}$ to the VBN with the formula $E(v) = \frac{size(v) * E_{ik}}{|Spec(S_{ik}, X)|}$.

The following theorem guarantees that propagation in the revised GenSpace graph has the same effect as in the original one.

Theorem 5.1 The linear GSEP method produces the same results for the original GenSpace graph and the virtual GenSpace graph in terms of potentially interesting nodes, if initial estimates in the original and virtual GenSpace graphs are consistent.

Proof.

(1) There are no zero old estimates.

First, we incorporate the VBN V in the original GenSpace graph. Set V as a child of bottom node X and the parent node of base interesting nodes. This will not change the results of the mining process.

When the user changes estimates in a potentially interesting node P, the estimates are first propagated to the bottom node, and then upward to the other nodes, including V.

Assume that node *P*'s estimates are changed. We propagate them to the bottom node *X*. For a section $x \in Spec(p, X)$ in the bottom node *X*, where *p* is a section of *P*, we have

$$E_{new}(x) = E_{old}(x)\frac{E_{new}(p)}{E_{old}(p)}.$$

When we propagate estimates upward to *V*, For an section $v \in V$ that satisfies $v \in Spec(p, V)$, we have

$$E_{new}(v) = \sum_{x \in Spec(v,X)} E_{new}(x) = \sum_{x \in Spec(v,X)} E_{old}(x)\frac{E_{new}(p)}{E_{old}(p)}.$$

Because initial estimates are consistent, i.e., $E_{old}(v) = \sum_{x \in Spec(v,X)} E_{old}(x)$, we have

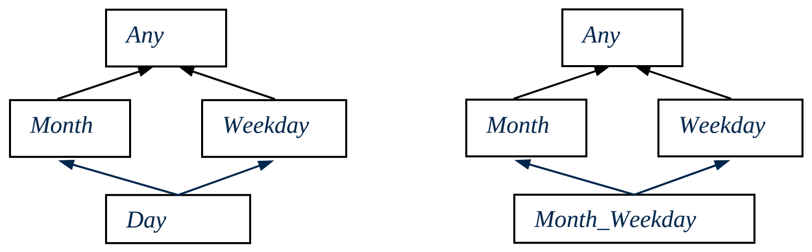
$$E_{new}(v) = E_{old}(v)\frac{E_{new}(p)}{E_{old}(p)}.$$

This new estimate for section *v* is identical to that obtained from the propagation in the virtual GenSpace graph. Then when we propagate the estimates upwards to all other potentially interesting nodes, we obtain the same results for estimates.

(2) There are zero old estimates.

Assume $E_{old}(p) = 0$. According to linear GSEP, we have $E_{new}(x) = \frac{E_{new}(p)}{Spec(p,\ X)}$ for the real bottom node. Then we propagate upward to the VBN, we obtain $E_{new}(v) = \sum_{x \in Spec(v,X)} E_{new}(x) = \frac{|\,Spec(v, X)\,|\,E_{new}(p)}{Spec(p,\ X)}$. This result is identical to that obtained from propagating from *P* to *V* in the virtual GenSpace graphs.

Example 5.1. Figure 5.2 illustrates the definition of a VBN. Figure 5.2(a) is an ExGen graph for a *Date* attribute for a specific non-leap year. Suppose that only the bottom node is specified as uninteresting and that all possible values occur in the data set. We have a domain of 365 elements in the bottom node *Day*, 12 sections for node *Month*, and 7 sections for node *Weekday*. The base interesting nodes are *Month* and *Weekday*. Figure 5.2(b) is the corresponding ExGen graph with a VBN *Month_Weekday*, with 12 * 7 = 84 sections {*January Sunday*, *January Monday*, …, *December Saturday*}. We can see that node *Month_Weekday* is a finer partition of the *Month* and *Weekday* nodes, and is a rougher partition of the *Day* node.

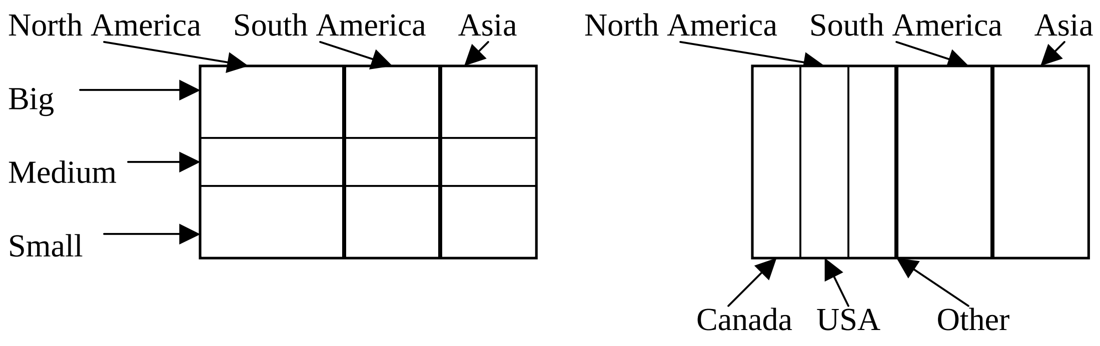


(a) Original ExGen Graph (b) ExGen Graph with a VBN

Figure 5.2. An example for VBN

Some factors influence the effect of the VBNs. First, the greater the generalization ratio is in the ExGen graphs, the better effect the VBN will achieve. This is because the size of the VBN is related to the size of the base interesting nodes.

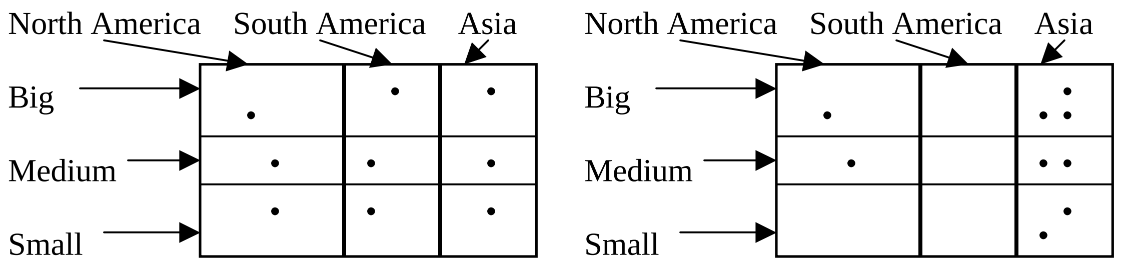
Secondly, the more the base interesting nodes are correlated, the better effect it will achieve. Assume that we have a set of countries in the Asia-Pacific region and there are two base interesting nodes (partitions) *A* and *B*. *A* = {*North America*, *South America*, *Asia*} is a partition based on continent. *B* = {*Big countries* (with a population more than 100 million people), *Medium countries* (with a population between 10 million and 100 million), *Small countries* (with a population less than 10 million)} is a partition based on population. They all partition the domain into three sections. Because there is no correlation between these two partitions, we get nine partitions when we combine them to construct the VBN as shown in Figure 5.3(a). The thick line denotes partition *A* and the thin line denotes the partition *B*. Then if we change the partition *B* to {*Canada*, *USA*, *Other countries*} and keep the partition *A* unchanged, the two partitions correlated in the way that two sections of partition *B* (*Canada* and *USA*) are subsets of a section of partition *A* (*North America*) as shown in Figure 5.4(b). In this case, we get five sections for the VBN.



(a) Partitions *A* and *B* are not correlated (b) Partitions *A* and *B* are correlated

Figure 5.3 The correlation among the base interesting nodes and the size of VBN

Thirdly, the more compact the data distribution is, the better effect it will bring. Based on the partition in Figure 5.3(a), if the country set is {*USA*, *Canada*, *Bermuda*, *Brazil*, *Argentina*, *Bolivia*, *China*, *Korea*, *Singapore*} the data are distributed in all sections in the VBN as shown in Figure 5.4. In the VBN, there will be nine sections. The VBN will not help reduce the storage. However, if the county set is {*USA*, *Canada*, *India*, *China*, *Japan*, *Korea*, *Thailand*, *Singapore*, *Nepal*}, the data are distributed in only five sections as shown in Figure 5.4(b). Therefore, in the VBN, storage cost will be reduced by nearly 50%.



(a) Data distributed in all sections. (b) Data distributed in only five sections.

Figure 5.4 Data distribution and the size of VBN

From the analysis, we can see that the size of a VBN cannot exceed the size of the real bottom node and the multiplication of the sizes of the basic interesting nodes, i.e., $|V| \leq \min(|X|, \prod_{B_i \in BI} |B_i|)$, where V denotes the VBN, X denotes the real bottom nodes, and B_i denotes a base interesting node. The lower bound of the size of the VBN is the maximum size of the base interesting nodes, i.e., $|V| \geq \max_{B_i \in BI} |B_i|$. This occurs when one of the base interesting nodes is a finer partition of all other base interesting nodes. Although the two partitions may not have a generalization relation logically, the relation can exist for a given subset of values. For example, we can partition a set of countries into P_1 = {*North America*, *Asia*, *South America*}. We can also partition them into P_2 = {*Canada*, *Other countries*}. Logically, there is no generalization relation between these two partitions. However, if we do not have North American countries except *Canada* in the data set, we can see that $P_1 \preceq P_2$. In this case, the VBN is identical to P_1. We can call this generalization relation a ***factual generalization relation***.

5.2 Experimental results

We conducted experiments on the Saskatchewan weather dataset and the University of Regina student data set to measure the effectiveness of using VBNs.

For the Saskatchewan weather data set, we assumed all bottom nodes in ExGen graphs are uninteresting. For the attributes *HighTemp* and *TotalPrep*, which have numeric values and have only one child node of the bottom node in their ExGen graphs, we simply eliminated the bottom node. For attribute *Station*, we replaced the bottom node *Specific-Station* with a VBN called *C-D-L-Region*. For attribute *Date*, we replaced the bottom node *Specific-Date* with a VBN called *YYYYMM-Season*. Table 5.1 lists the sizes of the real bottom node and the VBN for the ExGen graphs for the individual attributes and the Genspace graph formed by combining these ExGen graphs. Figure 5.5 shows the percentage of time cost saved by traversing in a virtual GenSpace graph compared with traversing in the original GenSpace graph. We can see that the percentage of the propagation time saved increases as the size of the bottom node increases. Figure 5.6 compares the time cost and space cost of virtual and real bottom nodes. The X-axis denotes the size of the bottom node, and Y-axis denotes the propagation cost and space cost in thousands of records, respectively. All costs increase linearly with the size of the bottom node, but the rate of increase is smaller for the VBN than for the real bottom node.

After incorporating VBNs in ExGen graphs, any nodes in the GenSpace graph involving the virtual values are not interesting to the user, because the virtual values are only intended to facilitate the propagation and they do not necessarily have any logical meaning. When we mark them as uninteresting nodes and apply the node pruning algorithm presented in Chapter 4, we can further reduce the time and storage costs. Figure 5.7 compares the costs.

Table 5.1 Comparison of the sizes of the real bottom nodes and VBNs for the weather data set

Size	Station	Date	HighTemp	TotalPrep	GenSpace
Bottom node	30	18262	152	280	211584
VBN	21	1200	4	4	43808

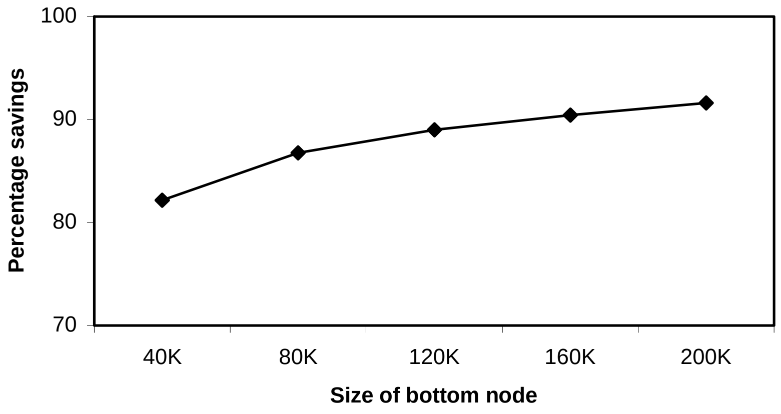
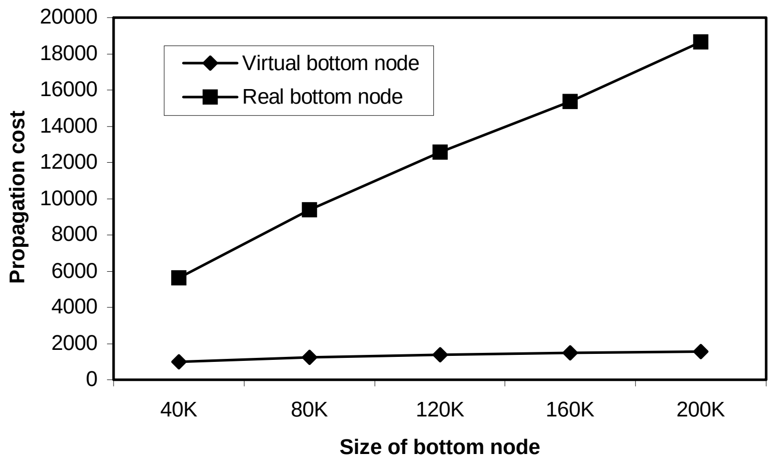
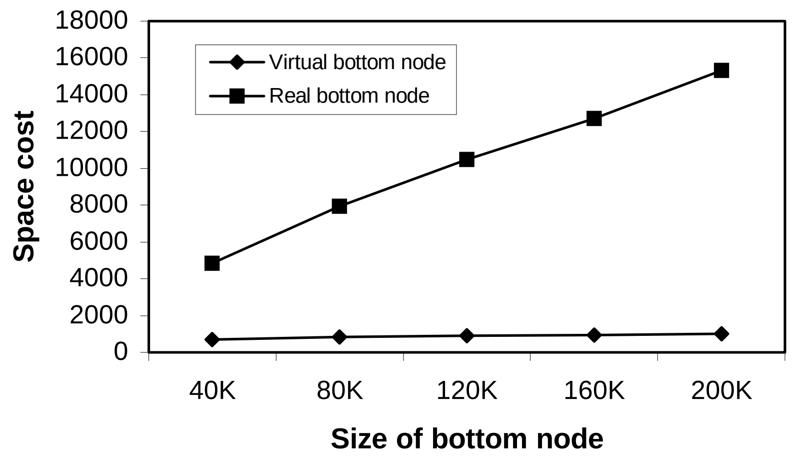


Figure 5.5 Percentage time savings for VBNs for the weather data set

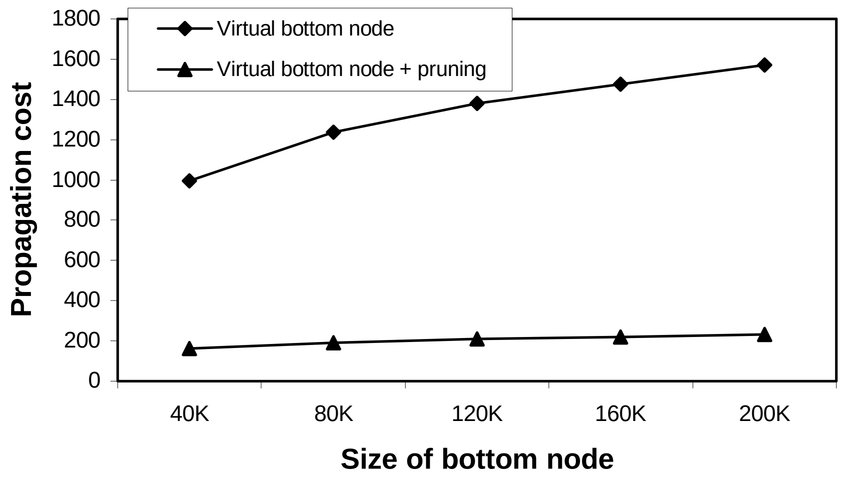


(a) Propagation cost

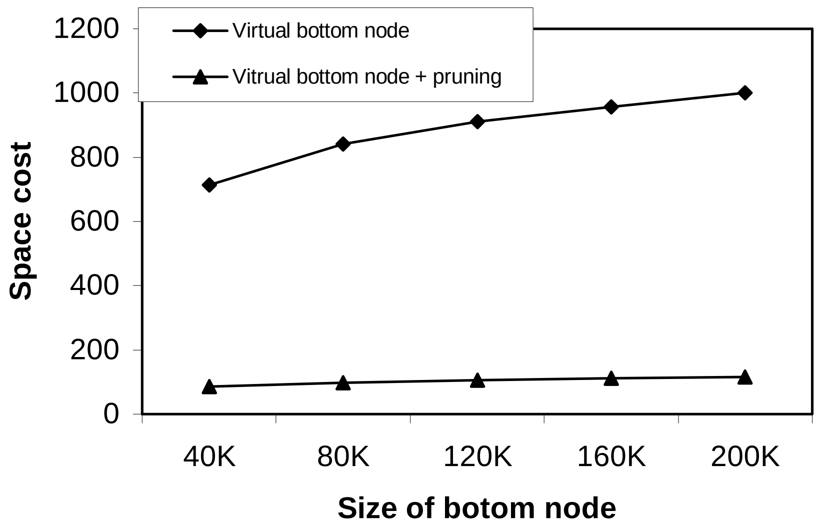


(b) Space cost

Figure 5.6 Time and storage costs for virtual and original GenSpace graphs for the weather data set



(a) Propagation cost



(b) Space cost

Figure 5.7 Time and storage cost for the virtual GenSpace graph and the virtual GenSpace subgraph for the weather data set

For the University of Regina student data set, we also assumed that all bottom nodes in ExGen graphs are uninteresting. Table 5.2 lists the size of the real bottom node and the VBN for the ExGen graphs for the individual attributes and the Genspace graph formed by combining these ExGen graphs. Figure 5.8 shows the percentage of time cost saved by traversing in a virtual GenSpace graph compared with traversing in the original GenSpace graph. We can see that the percentage of propagation time saved increases as the size of the bottom node increases. Figure 5.9 compares the time cost and space cost of virtual and real bottom nodes. All costs increase linearly with the size of the bottom node, but the rate of increase is smaller for the VBN than for the real bottom node. Figure 5.10 compares the time and the storage costs for traversing the GenSpace graph with those of traversing the pruned GenSpace graph. All the experiments on the student data set have similar results to those obtained from the weather data set.

Table 5.2 Comparison of the sizes of the real bottom nodes and the VBNs for the student data set

Size	City	Program	Term	GPA	GenSpace
Bottom node	832	159	6	62	59361
VBN	64	88	6	9	4599

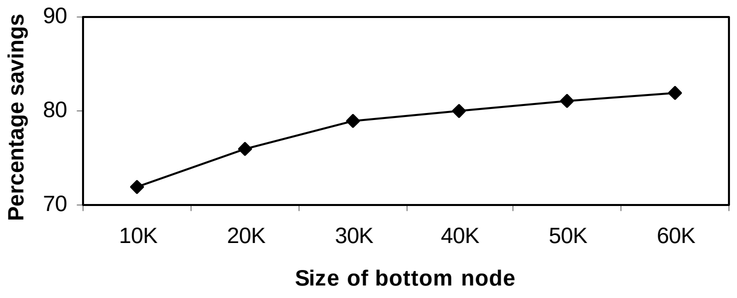
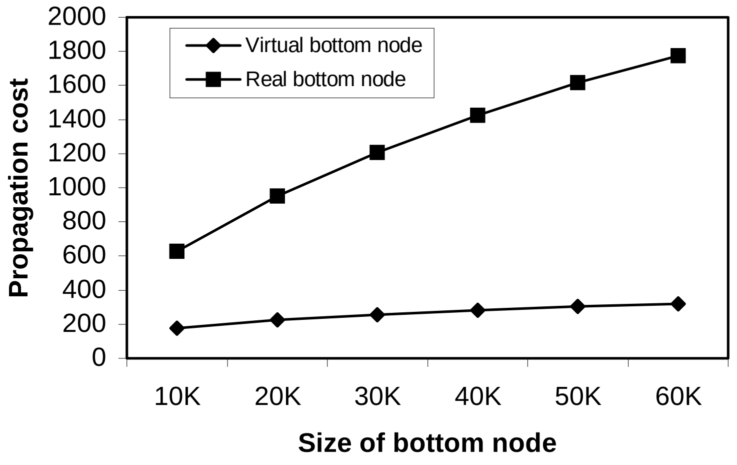
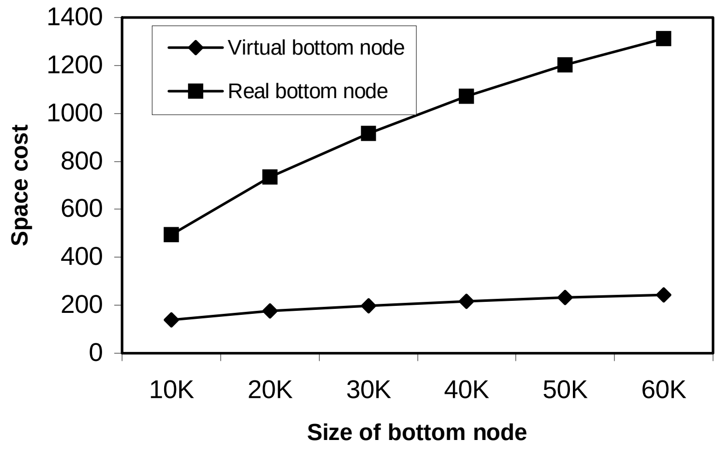


Figure 5.8 Percentage time savings for VBNs for the student data set

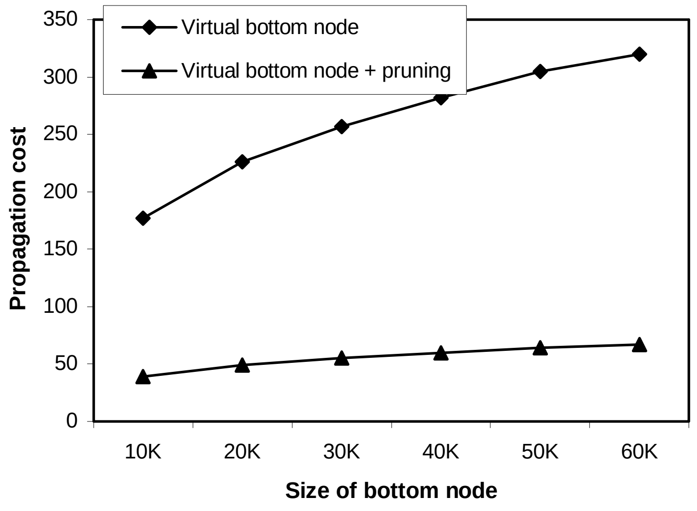


(a) Propagation cost

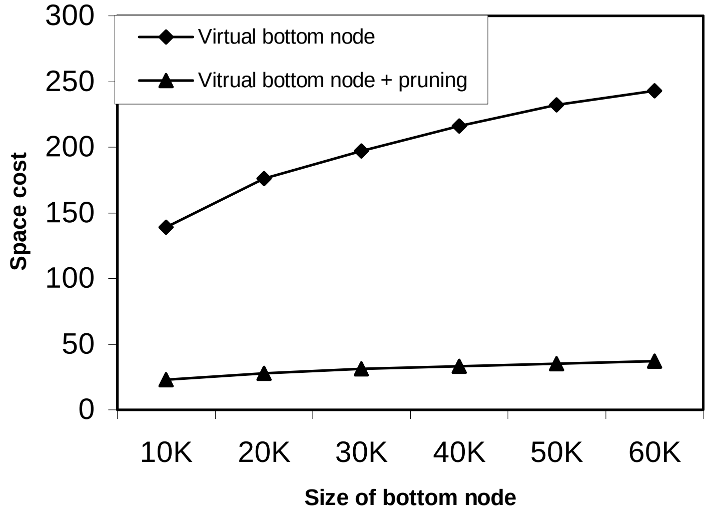


(b) Space cost

Figure 5.9 Time and storage cost for virtual and original GenSpace graphs for the student data set



(a) Propagation cost



(b) Space cost

Figure 5.10 Time and storage cost for the virtual GenSpace graph and the virtual GenSpace subgraph for the student data set

CHAPTER 6

INTERESTINGNESS MEASURES IN GENSPACE SUMMARY SELECTION

In this chapter, we analyze the interestingness measures in the GSSS process, which selects the summaries that are far from the user's estimates and presents them to the users. The key issue in the GSSS process is to choose the appropriate interestingness measures that measure the distance between the observed probability distributions and the estimates. Since the user's estimates are integrated in the measures in the GSSS process, the measures can be considered as subjective measures. In the remainder of this chapter, we first review 16 common objective interestingness measures for summaries and the principles for choosing the measures. We then choose the appropriate measures to convert to subjective measures for GSSS by incorporating the user's estimates. Finally we identify the properties of these measures in terms of the GSSS process.

6.1 Interestingness measures for summaries

Hilderman and Hamilton reviewed 16 objective interestingness measures for summaries [Hilderman and Hamilton, 2001] (see Table 2.4). These measures rank summaries according to the diversity of their distribution. They also proposed five principles desired for any acceptable interestingness measures for summaries. The formulas of the measures are listed in the second column in Table 2.4. Here we only consider the ***concentration order*** [Hilderman and Hamilton, 2001], which means that the further the distribution is from the uniform distribution, the higher value the interestingness measure will attain, and thus the more interesting the summary is.

These interestingness measures are widely used in various domains to measure the diversity distribution of populations, such as ecology [Bulla, 1994; Molinari, 1989; Peet, 1974], economy [Allison, 1978; Atkinson, 1970; Schutz, 1951], biology [Lewontin, 1972], and business [Attaran and Zwick, 1989; Hart, 1971]. Although the interestingness measures are highly dependent on the specific domain and there is not a single standard for choosing them, some fundamental principles have been proposed to evaluate the measures. They are ***Minimum Value Principle* (*P*1)**, ***Minimum Value Principle* (*P*2)**, ***Skewness Principle* (*P*3)**, ***Permutation Principle* (*P*4)**, and ***Transfer Principle* (*P*5)**. Hilderman and Hamilton identified relations between the measures and the principles as shown in Table 2.4 [Hilderman and Hamilton, 2001].

6.2 Incorporating Estimates in Interestingness Measures

In GSSS process, the interestingness measures need to incorporate the user's estimates. The most straightforward way to do so is to substitute the average probability $\bar{q}$ in measures in Table 2.4 with the estimates e_i. Since the Simpson measure, the Shannon measure, the Total measure, and the McIntosh measure do not contain $\bar{q}$, they are not suitable to incorporate the estimates. The Gini measure is also not straightforward to incorporate estimates. Therefore, we incorporate the estimates in the other nine measures and convert them to subjective measures as shown in Table 6.1. In Table 6.1, o_i and e_i denotes the observed probability and the estimate, respectively, for the *i*th record in the summary.

Here we distinguish these measures into ***distance measures*** (***D***), ***similarity measures*** (***S***), and ***other measures*** (***O***).

Definition 6.1 A ***distance measure*** d of observed probability distribution $o = [o_1, \ldots, o_n]$ and estimates $e = [e_1, \ldots, e_n]$ is a non-negative real valued function $d(o, e): R^n \times R^n \to [0, +\infty]$. It attains the minimum value 0 iff $o = e$ is satisfied.

The distance measures measure the distance between two vectors. When the two vectors are identical, the distance is zero. Farther distance results in higher interestingness values. Based on Definition 6.1, the variance measure, Schutz measure, Kullback-Liebler measure, and Theil measure are distance measures.

It must be noted that there are differences between our definition of distance and the traditional definition of distance in mathematics. The traditional definition of the distance function requires the symmetric property $d(x, y) = d(y, x)$ and triangular inequality $d(x, y) \leq d(x, z) + d(z, y)$ to be satisfied. Our definition of distance between the observed probabilities and the estimates is more relaxed than the traditional one.

Definition 6.2 A ***similarity measure*** of observed probability distribution $o = [o_1, \ldots, o_n]$ and estimates $e = [e_1, \ldots, e_n]$ is a function $s(o, e): R^n \times R^n \to [0, 1]$, which satisfies $s(o, e) = 1$ iff $o = e$.

The similarity measures measure the similarity of two vectors. Similarity value 0 corresponds to the maximum dissimilarity and 1 corresponds to the maximum similarity.

Any similarity measure can be converted to a distance measure. They are distinguished here because we need to know their semantics to explain the interestingness of summaries in GSSM process.

If a measure is neither a distance measure nor a similarity measure, we call it the other measure. The Type column in Table 6.1 classifies the measures according to this criterion. The reason that we classify the measures in this way is that we need to interpret the semantics of the interesting values in GSSS process. Since the unexpectedness is represented by the distance between the observed probabilities and the estimates, if a distance measure is used, the summaries should be ranked in the descending order. If a similarity measure is used, the summaries should be ranked in the ascending order. The semantics of the other measures is difficult to explain in the context of GSSS.

Table 6.1 Subjective interestingness measures with the user's estimates

Measure	Definition	Type	Pruning
Variance	$\frac{\sum_{i=1}^{m}(o_i - e_i)^2}{m-1}$	D	
Lorenz	$\sum_{i=1}^{m}(m-i+1)o_i e_i$	O	
Schutz	$\frac{1}{2m}\sum_{i=1}^{m}\frac{\lvert o_i - e_i \rvert}{e_i}$	D	Upper bound
Bray	$\sum_{i=1}^{m}\min(o_i, e_i)$	S	Monotonic
Whittaker	$1-\frac{\sum_{i=1}^{m}\lvert o_i - e_i \rvert}{2}$	S	Monotonic
Kullback-Liebler	$\sum_{i=1}^{m} o_i \log_2 \frac{o_i}{e_i}$	D	Upper bound
MacArthur	$-\sum_{i=1}^{m}\frac{o_i+e_i}{2}\log_2\frac{o_i+e_i}{2}-\frac{\sum_{i=1}^{m} o_i \log_2 o_i + \log_2 m}{2}$	O	
Theil	$\frac{\sum_{i=1}^{m}\lvert o_i \log_2 o_i - e_i \log_2 e_i \rvert}{m e_i}$	D	
Atkinson	$1-\prod_{i=1}^{m}\frac{o_i}{e_i}$	O	

6.3 Properties of the Interestingness Measures That Can Be Used for Pruning

The pruning methods proposed in the previous chapters are aimed at improving the efficiency of the GSEP process. In this section, we propose pruning strategies based on different interestingness measures in the GSSS process.

After the pre-process of pruning nodes in ExGen Graphs and GenSpace graph and the estimate propagation, we begin the GSSS process. Intuitively, if the difference between the observed probability distribution and the estimates in a node is small enough, the difference of its descendent nodes will not be large, and thus are not interesting and may be pruned in the GSSS process. However, this is not always true for all measures. For example, the variance measure is not a monotonic function and the higher levels of summaries might have higher interestingness values than the lower levels of summaries, therefore, the pruning cannot be conducted for this measure. In this section we study the properties of some interestingness measures and show how these properties can be used as pruning strategies.

6.3.1 Monotonicity

Monotonicity can be defined in different ways in different contexts. In the context of GenSpace graphs and GSSS processes, we define monotonicity in the following way.

Definition 6.3 In a GenSpace graph *G*, if for any two given nodes n_1 and n_2 satisfying $n_1 \prec n_2$, an interestingness measure *m* satisfies $m(n_1) \le m(n_2)$, we call the measure *m* ***monotonic***.

Theorem 6.1. The Bray measure is monotonic in GenSpace graphs.

Proof:

Assume we have two nodes n_1 and n_2 satisfying $n_1 \prec n_2$. Each section $s \in n_2$ corresponds to a set of sections $spec(s, n_1)$ in n_1, with $e(s) = \sum_{r \in spec(s,n_1)} e(r)$ and $o(s) = \sum_{r \in spec(s,n_1)} o(r)$.

We have

$$\begin{aligned}
&\min(e(s), o(s)) \\
&= \min(\sum_{r \in spec(s,n_1)} e(r), \sum_{r \in spec(s,n_1)} o(r)). \\
&\geq \sum_{r \in spec(s,n_1)} \min(e(r), o(r))
\end{aligned}$$

Therefore,

$$\begin{aligned}
&m(n_2) \\
&= \sum_{s \in n_2} \min(e(s), o(s)) \\
&= \sum_{s \in n_2} \min(\sum_{r \in spec(s,n_1)} e(r), \sum_{r \in spec(s,n_1)} o(r)). \\
&\geq \sum_{s \in n_2} \sum_{r \in spec(s,n_1)} \min(e(r), o(r)) \\
&= m(n_1)
\end{aligned}$$

Theorem 6.2 The Whittaker measure is monotonic in GenSpace graphs.

Proof:

Assume we have two nodes n_1 and n_2 satisfying $n_1 \prec n_2$. Each section $s \in n_2$ corresponds to a set of sections $spec(s, n_1)$ in n_1, with $e(s) = \sum_{r \in spec(s,n_1)} e(r)$ and $o(s) = \sum_{r \in spec(s,n_1)} o(r)$.

We have

$$\begin{aligned}
&\frac{|e(s) - o(s)|}{2} \\
&= \frac{1}{2} | \sum_{r \in spec(s,n_1)} e(r) - \sum_{r \in spec(s,n_1)} o(r) | \\
&= \frac{1}{2} | \sum_{r \in spec(s,n_1)} (e(r) - o(r)) | \\
&\leq \frac{1}{2} \sum_{r \in spec(s,n_1)} | e(r) - o(r) |
\end{aligned}$$

Therefore,

$$
\begin{aligned}
& m(n_2) \\
& = 1 - \frac{1}{2} \sum_{s \in n_2} | e(s) - o(s) | \\
& \geq 1 - \frac{1}{2} \sum_{s \in n_2} \sum_{r \in spec(s, n_1)} | e(r) - o(r) | \\
& = m(n_1)
\end{aligned}
.
$$

Theorems 6.1 and 6.2 state that more general nodes always attain higher interestingness value than more specific ones. It also implies that the top node always obtains the maximum interestingness value and the bottom node always obtains the minimum interestingness value regardless of the distributions of o_i and e_i. Since both the Bray and the Whittaker measures are similarity measures, we always obtain the bottom nodes as the most interesting nodes.

The monotonicity property of the Bray and Whittaker measures can be used to prune the summaries when the user sets a threshold for the interestingness values and only wants to output the summaries that are below the threshold. Once the interestingness values of a node is above the threshold, all its decedents can be pruned.

6.3.2 Upper Bound

Some interestingness measures do not have the monotonicity property, but they still have an upper bound for the interestingness values for the descendent nodes of a given node. These measures include the Schutz measure, the symmetric relative variance measure, and the Kullback-Liebler measure.

6.3.2.1 Schutz Measure

The Schutz measure represents the relative difference of the observed probabilities and the estimates. Although it does not possess the monotonicity property, we identified a relaxed upper bound for the descents of a node. Lemma 6.1 shows this property.

Lemma 6.1 In a GenSpace graph, a node P consists of partition $\{S_1, S_2, \ldots, S_n\}$, and we use the Schutz as the interestingness measure. If for every partition S_i we have $d(S_i) = \frac{|e(S_i) - o(S_i)|}{e(S_i)} \le t, t \ge 0$, the interestingness values of all descendents of P are not greater than t.

Proof:

Because $\frac{|e(S_i) - o(S_i)|}{e(S_i)} \le t$, for any i, we have relative variance for node P

$$Intr(P) = \frac{1}{n}\sum_{i=1}^{n}\frac{|e(S_i) - o(S_i)|}{e(S_i)} \le t .$$

Suppose node Q is a descendent of node P. We have the partition for $Q = \{T_1, T_2, \ldots, T_m\}$, where $T_i = \bigcup_{S_j \in Spec(T_i, P)} S_j$.

Therefore,

$$\frac{|e(T_i)-o(T_i)|}{e(T_i)}$$
$$=\frac{|\sum_{S_j \in Spec(T_i,P)_i} e(S_j) - \sum_{S_j \in Spec(T_i,P)} o(S_j)|}{\sum_{S_j \in Spec(T_i,P)} e(S_j)}$$
$$=\frac{|\sum_{S_j \in Spec(T_i,P)} (e(S_j)-o(S_j))|}{\sum_{S_j \in Spec(T_i,P)} e(S_j)}$$
$$\leq \frac{\sum_{S_j \in Spec(T_i,P)} |e(S_j)-o(S_j)|}{\sum_{S_j \in Spec(T_i,P)} e(S_j)}$$

Because for any nonnegative real value x_i and y_i, we have $\frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} y_i} \leq \max(\frac{x_i}{y_i})$,

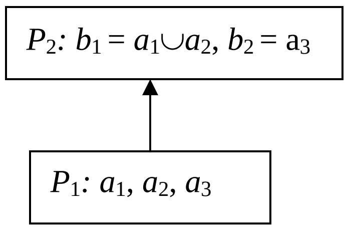
We get $\frac{|e(T_i)-o(T_i)|}{e(T_i)} \leq \max_{S_j \in Spec(T_i,P)} (\frac{|e(S_j)-o(S_j)|}{e(S_j)}) \leq t$,

Therefore, $Intr(Q)=\frac{1}{m}\sum_{i=1}^{n}\frac{|e(T_i)-o(T_i)|}{e(T_i)} \leq t$.

Theorem 6.3 shows that if all the relative differences of every section in a node have an upper bound, then the interestingness measures of its descendents cannot exceed this upper bound.

Here we define $d(S_i)=0$ if $e(S_i)=0$ and $o(S_i)=0$. We define $d(S_i)=\textit{the greatest value represented in the computer}$ if $e(S_i)=0$ and $o(S_i) \neq e(S_i)$.

Theorem 6.3 indicates that when we use the Schutz measure, the child's greatest relative difference is not greater than the parent's greatest relative difference. But it does not mean that the interestingness value of the child cannot be greater than that of its parent, as is demonstrated by the following example.



$o(a_1) = 0.3$, $e(a_1) = 0.4$
$o(a_2) = 0.3$, $e(a_2) = 0.4$
$o(a_3) = 0.4$, $e(a_3) = 0.2$

$o(b_1) = 0.6$, $e(b_1) = 0.8$
$o(b_2) = 0.4$, $e(b_2) = 0.2$

(a) GenSpace graph (b) Estimates and observed probability for nodes P_1 and P_2

Figure 6.1 An example to illustrate Lemma 6.1

In Figure 6.1(a), node P_1 is a parent of node P_2. P_1 has three sections a_1, a_2 and a_3. P_2 has two sections b_1 and b_2, where b_1 is the union of a_1 and a_2, and b_2 is identical to a_3. The observations and estimates for the two nodes are listed in Figure 6.1(b). We obtain $d(a_1) = \frac{|e(a_1) - o(a_1)|}{o(a_1)} = 0.33$; similarly, we have $d(a_2) = 0.33$, $d(a_3) = 0.5$, $d(b_1) = 0.33$, $d(b_2) = 0.5$, $Intr(P_1) = 1 / 2 * (0.33 + 0.5) = 0.42$, $Intr(P_2) = 1 / 3 * (0.33 + 0.33 + 0.5) = 0.37$. Although the child's interestingness value (0.42) is greater than that of its parent (0.37), it is not greater than the greatest relative difference of its parent node (0.5). If $t = 0.5$ has been set as the threshold, since all the differences of sections in P_1 is less than t, according to Lemma 6.1, the relative variances of the descendents of P_1 should be less than t, thus, the descendents of P_1 can be pruned.

6.3.2.2 Symmetric Relative Variance

We define a variant of the Schutz measure, called the ***symmetric relative variance***, as $Intr(P) = \frac{1}{n}\sum_{i=1}^{n} \frac{|e(S_i) - o(S_i)|}{\max(e(S_i), o(S_i))}$, where $o(S_i)$ and $e(S_i)$ are observed and estimated probabilities for section S_i, respectively.

Compared with the Schutz measure, the symmetric relative variance is symmetric for observed probabilities and the estimates and takes values in [0,1]. We found that the symmetric relative variance also has the upper bound property, which is presented in Lemma 6.2.

Lemma 6.2 In a GenSpace graph, a node P consists of partition $\{S_1, S_2, \ldots, S_n\}$, and we use the symmetric relative variance. If for all partition a_i we have $\frac{|e(S_i) - o(S_i)|}{\max(e(S_i), o(S_i))} \le t, t \ge 0,$ the interestingness values of all the descendents of P are not greater than t.

Proof:

To prove Lemma 6.2, we only need to prove that the relative difference for every section of a descendent node cannot be greater than t.

Let i be the number of the sections in node P that belong to a section in its descendent. We use induction on i for the proof.

(1) If $i = 2$, we assume section b in a descendent node corresponds to sections a_1 and a_2 in the node P, i.e., $b = a_1 \cup a_2$,

$$\frac{|e(b) - o(b)|}{\max(e(b), o(b))} = \frac{|e(a_1) + e(a_2) - o(a_1) - o(a_2)|}{\max(e(a_1) + e(a_2), o(a_1) + o(a_2))}$$

(1.1) If $e(a_1) \ge o(a_1)$ *and* $e(a_2) \ge o(a_2)$,

$$\frac{|e(b)-o(b)|}{\max(e(b),o(b))}=\frac{e(a_1)-o(a_1)+e(a_2)-o(a_2)}{e(a_1)+e(a_2)}$$

$$\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))}=\frac{e(a_1)-o(a_1)}{e(a_1)},\ \frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))}=\frac{e(a_2)-o(a_2)}{e(a_2)},$$

Therefore, we have $\frac{|e(b)-o(b)|}{\max(e(b),o(b))}\le \max(\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))},\frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))})$

(1.2) If $e(a_1)\le o(a_1)$ *and* $e(a_2)\le o(a_2)$, the proof is similar to the case (1.1).

(1.3) If $e(a_1)\ge o(a_1)$ *and* $e(a_2)\le o(a_2)$,

(1.3.1) If $e(a_1)+e(a_2)\ge o(a_1)+o(a_2)$,

we have $\frac{|e(b)-o(b)|}{\max(e(b),o(b))}=\frac{e(a_1)+e(a_2)-o(a_1)-o(a_2)}{e(a_1)+e(a_2)}$,

$$\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))}=\frac{e(a_1)-o(a_1)}{e(a_1)}$$

$$\begin{aligned}&\frac{|e(b)-o(b)|}{\max(e(b),o(b))}-\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))}\\&=\frac{e(a_1)+e(a_2)-o(a_1)-o(a_2)}{e(a_1)(e(a_1)+e(a_2))}\\&=\frac{e(a_2)o(a_1)-e(a_1)o(a_2)}{e(a_1)(e(a_1)+e(a_2))}\\&\le 0\end{aligned}$$

Therefore,

$$\begin{aligned}&\frac{|e(b)-o(b)|}{\max(e(b),o(b))}\\&\le\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))}\\&\le\max(\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))},\frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))})\end{aligned}$$

(1.3.2) If $e(a_1)+e(a_2)\le o(a_1)+o(a_2)$, similarly we can get

$$\frac{|e(b)-o(b)|}{\max(e(b),o(b))}-\frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))}=\frac{e(a_2)o(a_1)-e(a_1)o(a_2)}{o(a_1)(o(a_1)+o(a_2))}\le 0,$$

$$\frac{|e(b)-o(b)|}{\max(e(b),o(b))}\le\frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))}\le\max(\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))},\frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))})$$

(1.4) If $e(a_1)\le o(a_1)$ *and* $e(a_2)\ge o(a_2)$, the proof is similar to situation (1.3).

Therefore, for i = 2, we have

$$\frac{|e(b)-o(b)|}{\max(e(b),o(b))}\le\max(\frac{|e(a_1)-o(a_1)|}{\max(e(a_1),o(a_1))},\frac{|e(a_2)-o(a_2)|}{\max(e(a_2),o(a_2))})$$

(2). Assuming for $i = k$,

we have $\frac{|e(c)-o(c)|}{\max(e(c),o(c))}\le\max_{i\in[1,k]}(\frac{|e(a_i)-o(a_i)|}{\max(e(a_i),o(a_i))})$.

When $i = k + 1$,

$$\begin{aligned}\frac{|e(b)-o(b)|}{\max(e(b),o(b))}&=\frac{|e(c)+e(a_{k+1})-o(c)-o(a_{k+1})|}{\max(e(c)+e(a_{k+1}),o(c)+o(a_{k+1}))}\\&\le\max(\frac{|e(c)-o(c)|}{\max(e(c),o(c))},\frac{|e(a_{k+1})-o(a_{k+1})|}{\max(e(a_{k+1}),o(a_{k+1}))})\\&\le\max(\max_{i\in[1,k]}(\frac{|e(a_i)-o(a_i)|}{\max(e(a_i),o(a_i))}),\frac{|e(a_{k+1})-o(a_{k+1})|}{\max(e(a_{k+1}),o(a_{k+1}))})\\&=\max_{i\in[1,k+1]}(\frac{|e(a_i)-o(a_i)|}{\max(e(a_i),o(a_i))})\end{aligned}$$

6.3.2.3 Kullback-Liebler measure

The Kullback-Liebler interestingness measure $Intr(P)=\sum_{i=1}^{n}o(S_k)\log_2(\frac{o(S_k)}{e(S_k)})$ is based on information theory. We also identified the upper bound for this measure.

Lemma 6.3 In a GenSpace graph, a node P consists of partition $\{S_1, S_2, \ldots, S_n\}$, and we use the Kullback-Liebler interestingness measure. If for every partition S_i we have

$\log_2(\frac{o(S_k)}{e(S_k)}) \leq m$, the interestingness values of all descendents of *P* are not greater than *m*.

Proof:

Suppose node *Q* is a descendent of node *P*. We have the partition for *Q* = {T_1, T_2, …, T_m}, where $T_i = \bigcup_{S_j \in Spec(T_i,\, P)} S_j$,

$$\log_2 \frac{o(T_i)}{e(T_i)} = \log_2 \frac{\sum_{S_j \in Spec(T_i,\, P)} o(S_j)}{\sum_{S_j \in Spec(T_i,\, P)} e(S_j)}$$

Because for any nonnegative real value x_i and y_i, we have $\frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} y_i} \leq \max(\frac{x_i}{y_i})$, and

$f(x) = \log_2 x$ is an increasing function,

We get $\log_2 \frac{o(T_i)}{e(T_i)} = \log_2 \frac{\sum_{S_j \in Spec(T_i,\, P)} o(S_j)}{\sum_{S_j \in Spec(T_i,\, P)} e(S_j)} \leq \log_2(\max_{S_j \in Spec(T_i,\, P)} \frac{o(S_j)}{e(S_j)}) \leq m$,

Therefore, $Intr(Q) = \sum_{i=1}^{n} e(T_i) \log_2 \frac{o(T_i)}{e(T_i)} \leq \sum_{i=1}^{n} e(T_i) m = m$.

6.3.2.4 Pruning Based on Upper Bound Property

Figure 6.2 illustrates a pruning strategy based on the upper bound property. This search algorithm finds the *k* most interesting summaries. In a run, if the user accepted the estimates for the observed probability, i.e., the estimates and observed probabilities are identical, all its descendants will not be interesting and they will be pruned. This is a special case of our heuristic.

In the thesis, we present the GSEP and GSSS as two separate steps. In implementation, we combined these two steps, because some nodes can be pruned based on interestingness measures in GSSS process and thus estimates do not need to be propagated to these nodes in GSEP process.

```
GSProp Algorithm
Input: a GenSpace graph $G$, $k$ (find top $k$ nodes)
1. From bottom to top in $G$, select the first $k$ nodes using breadth first search and mark them as the visited nodes.
2. Calculate the interestingness values for the selected nodes.
2.3. Let $t$ be the smallest interestingness value in the $k$ nodes.
4. For the next unvisited and unpruned node in the breadth first
     search,
5.        Calculate the difference between observations and estimates
          for each section.
6.        If the differences are all less than $t$,
7.                Prune this node and all its descendant nodes,
8.        Otherwise,
9.                Calculate its interestingness measure.
10.               If the interestingness measure is greater than $t$,
11.                       Replace the node with the least
                          interestingness value in the set of selected
                          nodes with the new node.
12.                       Set $t$ to be the least interestingness value of
                          the currently selected nodes.
```

Figure 6.2 Pruning algorithm GSProp for searching the GenSpace graph

CHAPTER 7

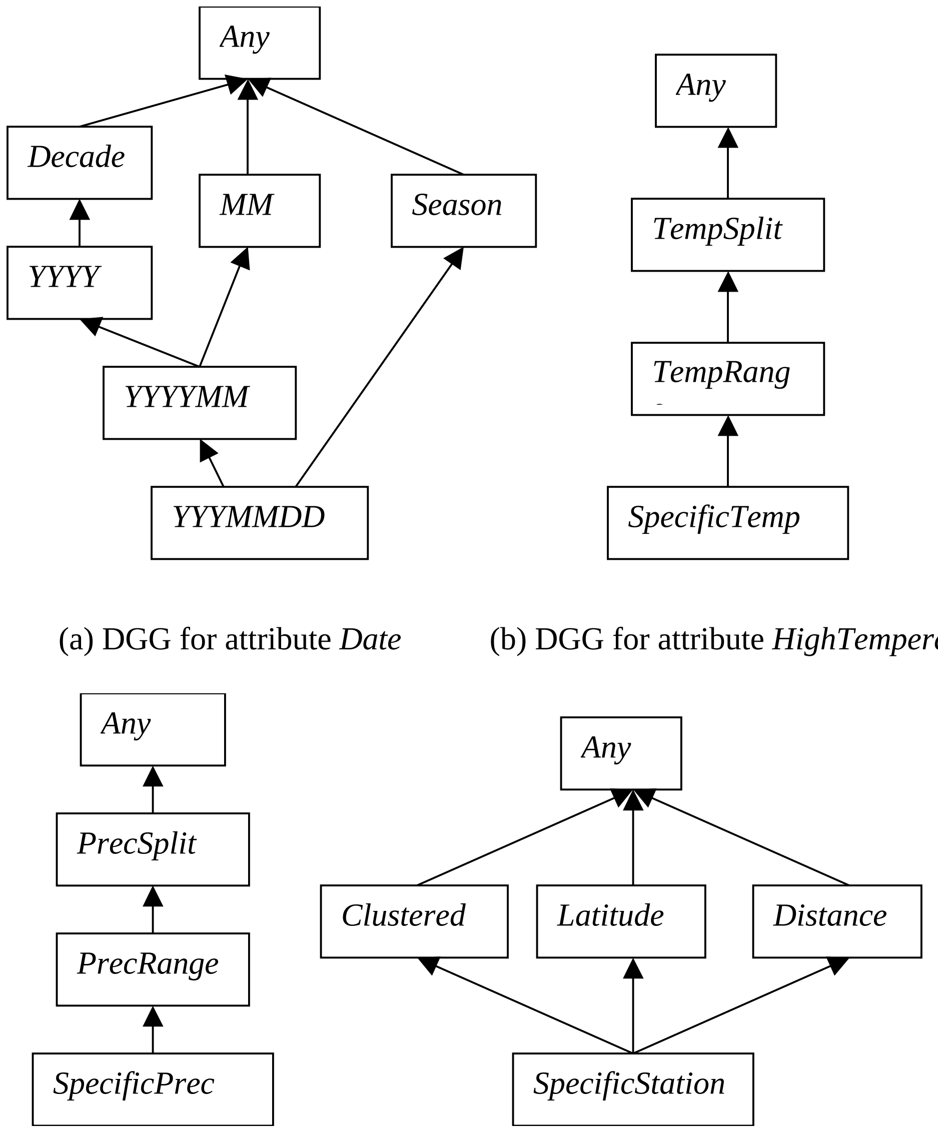
APPLICATIONS

In this Chapter, we demonstrate the effectiveness of the GSSM method by applying it to three real data sets, the Saskatchewan weather data set, the University of Regina student data set, and a customer data set. We will demonstrate the mining process as well as analyze the mining results.

7.1 Saskatchewan Weather Data Set

The Saskatchewan weather data set contains 211,534 records concerning daily weather observations in Saskatchewan from January 1, 1900 to December 31, 1949. Our goal was to assess whether, with no previous experience with this dataset, the GSSM discovery methodology could guide exploration of the data.

We used four attributes, *Station, Date, HighTemperature* (temperature in Celsius), and *TotalPrecip* (precipitation in mm), in our experiments. Attribute *Date* has format *YYYYMMDD*, including the information of year, month, and day. We generalize it to six nodes, *YYYYMM* (year and month), *MM* (month), *YYYY* (year), Decade, Season, and *Any* (any time). *High Temperature* was generalized to three nodes: *TempRange* with the domain {*cold*, *cool*, *warm*, *hot*}, *TempSplit* with the domain {high, low}, and *Any*. *TotalPrecip* was generalized to three nodes: *PrepRange* with the domain {0, (0, 50], (50, 100], (100, 400], >400}, *PrepSplit* with the domain {*no*, *yes*}, and *Any*. *Station* was generalized to four nodes according to geographic features and node *Any*. Figure 7.1 shows the DGGs for these attributes. The three interior nodes in the *Station* DGG correspond to the three clustering maps shown in Figure 7.2.



(a) DGG for attribute *Date* (b) DGG for attribute *HighTemperature*

(c) DGG for attribute *TotalPrecip* (d) DGG for attribute *Station*

Figure 7.1 DGGs for *Date*, *HighTemperature*, *TotalPrecip*, and *Station* attributes

Figure 7.2(a) shows the stations clustered using the *k*-means algorithm with $k = 6$ (other values of *k* gave less plausible maps). Figure 7.2(b) shows ten regions defined based on the adjusted distance *d* to the southwest corner of the province, which is at

(49°N, 110°E), using the formula

d = (Latitude – 49) + 0.35 (110 – Longitude).

This division assumes that similar weather occurs along a slanted line across the province. The 0.35 is a constant defined based on a map of the province showing ecological regions [Ecoregions]. Since the northeast corner of the province is at (60°N, 102°E), the values for d ranged from 0 to 13.8. To create the *DistanceRegion* node, we divided this range into 10 equal-sized intervals. Figure 7.2(c) shows the stations grouped from south to north into four regions (*South*, *LowMid*, *HighMid*, and *North*) to create the Latitude node.

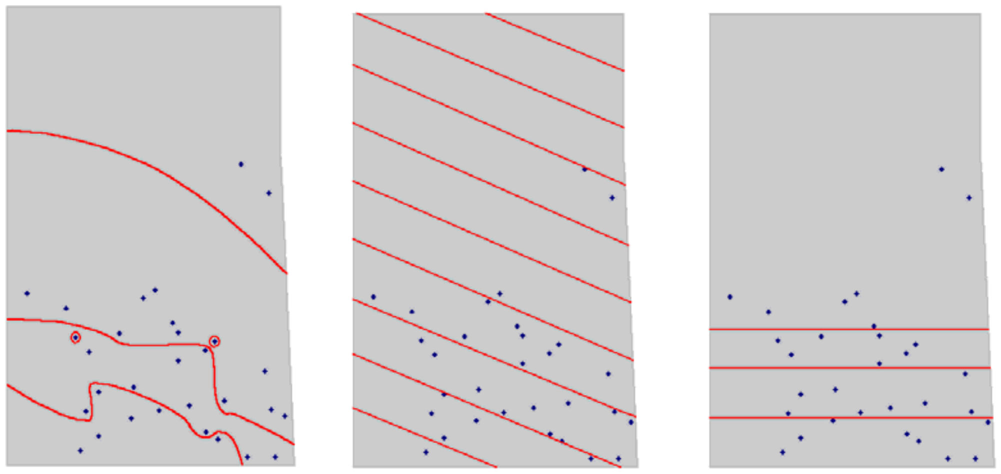

(a) Clustered Regions (CRegion) (b) Distance Regions (DRegion) (c) Latitude Regions (LRegion)

Figure 7.2 Maps corresponding to the interior DGG nodes for the *Station* Attribute

Associate initial estimates: In the beginning of the exploration process, we assumed initial estimates for some nodes in the DGGs. For the *Date* attribute, we assumed a uniform distribution at the *Year* (*YYYY*) node, i.e., we assumed an equal number of observations from each year 1900-1949. For the *Station* attribute, we assumed a uniform distribution at each weather station (*SpecificStation* node). For the *HighTemperature* attribute, we assumed a uniform distribution at the *TempRange* node, which means that cold days, cool days, warm days, and hot days all have probability value of 0.25. Similarly, for *TotalPrecip*, we assumed a uniform distribution at the *PrecRange* node, which means that the numbers of days for no precipitation, low precipitation, medium precipitation, and high precipitation are equal.

Generalize the data: We generated the framework of the GenSpace graph from DGGs without materialization. Since the *Date*, *HighTemperature*, *TotalPrecip*, and *Station* DGGs have 7, 4, 4, and 5 nodes, respectively, the number of aggregations or summaries produced is $7 \times 4 \times 4 \times 5 - 1 = 559$. We set the summaries that involve the specific values for *Date*, *HighTemperature*, and *TotalPrecip* attributes as uninteresting. Then the weather data are aggregated in all possible ways consistent with the DGGs according to the optimal tree in the GenSpace subgraph. We assume that the four attributes are independent and obtain the initial estimates for the node (*SpecificStation*, *YYYY*, *TempRange*, *PrecRange*) in the GenSpace graph. The system then propagates them to all the other nodes.

Rank the summaries: We used the variance measure for ranking the summaries in the experiments.

Table 7.1 shows the top 10 summaries for the first iteration. To get more insight into the highest ranked summaries, we use the chi-square test to test the fitness of the observations and estimates and determine which detailed summary records should be provided to the user. In this case, the highest ranked summary, which corresponds to the (*Any*, *Any*, *Any*, *PrecSplit*) node combination, is automatically translated into the English sentence “More readings (79%) have precipitation = NONE than estimated (50%).” This summary shows that, when Station, Date, and Temperature are ignored (set to *Any*), the percentage of daily observations in the data with rain is 21%, and that without is 79%. Since the original estimate was 50%/50%, according to the variance measure, the observed distribution of this summary is the farthest from its estimated distribution among all the summaries in the GenSpace subgraph.

Adjust estimates: At this point, the user has learned something about the domain, and the estimates can be adjusted according to the acquired knowledge. To continue this example, we assume that the distribution is simply accepted for the (*Any*, *Any*, *Any*, *PrecipSplit*) node and propagated to all the other nodes in the GenSpace graph. This assumption follows in a straightforward fashion from the results, and can be readily automated. The effect on further data mining corresponds to saying: “I accept that only 21% of the days have precipitation; now, don't tell me about that again or about any logical consequence of that.”

Table 7.1 Top 10 summaries after the first iteration

Station	**Date**	**Temperature**	**Precipitation**	**Variance**
Any	Any	Any	PrecSplit	6.88e-1
Any	Any	TempSplit	PrecSplit	1.16e-1
Any	Any	Any	PrecRange	1.11e-1
LRegion	Any	Any	PrecSplit	2.63e-2
Any	Any	TempRange	PrecSplit	2.55e-2
Any	Any	TempSplit	PrecRange	2.51e-2
Any	Season	Any	PrecSplit	2.473-2
CRegion	Any	Any	PrecSplit	2.07e-2
DRegion	Any	Any	PrecSplit	2.02e-2
Any	Decade	Any	PrecSplit	1.81e-2

Continue data mining: After revising the estimates and thereafter propagating them and calculating the variances of the summaries, we obtain the 10 most interesting summaries in Table 7.2. Most summaries with *PrecipSplit* or *PrecipRange* have disappeared from the top ten list because the changes of the estimates of *PrecipSplit* affect those of *PrecipRange* appropriately. The summaries (*Any*, *Any*, *Any*, *PrecRange*) and (*Any*, *Decade*, *Any*, *PrecSplit*) remain in the top ten. However, the summary (*Any*, *Any*, *Any*, *PrecRange*) ranks seventh instead of third, and its interestingness measure decreased from 1.11e-1 to 2.91e-3. Although summary (*Any*, *Decade*, *Any*, *PrecSplit*) has a higher ranking (from 10 to 8), its interestingness measure decreased from 1.81e-2 to 2.87e-3. This indicates that through estimate propagation from node (*Any*, *Any*, *Any*, *PrecSplit*), the distribution for other nodes in this GenSpace graph has been adjusted appropriately.

Now, the highest ranked summary tells us that the estimates for *Decade* node are the farthest from the observed ones. Among the five decades from 1900 to 1949, the percentage of observations from the decades in order are: 7%, 13%, 21%, 25%, and 34%. Again, the user can simply accept this, or explore deeper reasons. The actual reason was that only a few weather stations existed in 1900 and others were gradually added. We simply accept the observed distribution as the new estimates for *Decade* and propagate it to all the other nodes.

Table 7.2 Top 10 summaries after the second iteration

Station	**Date**	**Temperature**	**Precipitation**	**Variance**
Any	Decade	Any	Any	1.05e-2
Any	Any	TempSplit	Any	8.42e-3
Any	Season	TempSplit	Any	6.70e-3
LRegion	Any	Any	Any	6.34e-3
CRegion	Any	Any	Any	5.88e-3
Any	Any	TempRange	Any	3.77e-3
Any	Any	Any	PrecRange	2.91e-3
Any	Decade	Any	PrecSplit	2.87e-3
Any	Decade	TempSplit	Any	2.54e-3
DRegion	Any	Any	Any	2.38e-3

Tables 7.3 to 7.6 list the ten most interesting summaries for the third to sixth

iterations.

When the top-ranked summary has more than one domain value that is not "*Any*", the summary corresponds to a ***joint probability distribution***. For example, after the fourth iteration, the top-ranked summary is (*Any*, *Season*, *TempSplit*, *Any*), which means that the proportion of *Low Temperature* and *High Temperature* days varies with the season. To accept this distribution, a joint estimate between *Season* and *TempSplit* is specified. After propagation of the joint estimate and calculation of the interestingness of the summary, we can see that not only has (*Any*, *Season*, *TempSplit*, *Any*) disappeared from the top ten summaries, but also closely related nodes (*Any*, *Season*, *TempSplit*, *PrecSplit*) and (*Any*, *Season*, *TempRange*, *Any*) have become less interesting and also disappeared.

Table 7.3 Top 10 summaries after the third iteration

Station	**Date**	**Temperature**	**Precipitation**	**Variance**
Any	Any	TempSplit	Any	8.42e-3
Any	Season	TempSplit	Any	6.70e-3
LRegion	Any	Any	Any	6.34e-3
CRegion	Any	Any	Any	5.88e-3
Any	Any	TempRange	Any	3.77e-3
Any	Any	Any	PrecRange	2.91e-3
DRegion	Any	Any	Any	2.38e-3
Any	Season	TempSplit	PrecSplit	2.11e-3
Any	Season	TempRange	Any	2.07e-3
Any	Any	TempSplit	PrecSplit	1.92e-3

Table 7.4 Top 10 summaries after the fourth iteration

Station	Date	Temperature	Precipitation	Variance
Any	Season	TempSplit	Any	6.40e-3
LRegion	Any	Any	Any	6.34e-3
CRegion	Any	Any	Any	5.88e-3
Any	Any	Any	PrecRange	2.91e-3
CRegion	Any	Any	PrecSplit	1.76e-3
DRegion	Any	Any	Any	2.38e-3
Any	Any	TempRange	Any	2.36e-3
Any	Season	TempSplit	PrecSplit	2.01e-3
Any	Season	TempRange	Any	2.00e-3
LRegion	Any	Any	PrecSplit	1.72e-3

Table 7.5 Top 10 summaries after the fifth iteration

Station	Date	Temperature	Precipitation	Variance
LRegion	Any	Any	Any	6.34e-3
CRegion	Any	Any	Any	5.88e-3
Any	Any	Any	PrecRange	5.29e-3
DRegion	Any	Any	Any	2.388e-3
Any	Any	TempRange	Any	2.36e-3
CRegion	Any	Any	PrecSplit	1.76e-3
LRegion	Any	Any	PrecSplit	1.72e-3
LRegion	Any	TempSplit	Any	1.50e-3
CRegion	Any	TempSplit	Any	1.46e-3
CRegion	Any	Any	PrecRange	7.73e-4

Table 7.6 Top 10 summaries after the sixth iteration

Station	Date	Temperature	Precipitation	Variance
CRegion	Any	Any	Any	3.99e-3
Any	Any	Any	PrecRange	2.91e-3
Any	Any	TempRange	Any	2.36e-3
DRegion	Any	Any	Any	1.35e-3
CRegion	Any	Any	PrecSplit	1.20e-3
CRegion	Any	TempSplit	Any	1.01e-3
Any	Any	TempSplit	PrecRange	6.69e-4
Any	Any	TempRange	PrecSplit	5.87e-4
CRegion	Any	Any	PrecRange	5.70e-4
Any	Season	TempRange	Any	5.15e-4

To verify the observation that after accepting the observed probability distribution of the top summary and propagating it to the other summaries, its closely related summaries become less interesting, we did the following analysis.

We first define the ***similarity between two nodes n_1 and n_2*** in a DGG for attribute *a*. If $n_1 \le n_2$ or $n_2 \le n_1$, we define $similarity_a(n_1, n_2) = \alpha^n$ where $\alpha \in [0, 1]$ is a constant value and *n* denotes the minimum distance between n_1 and n_2 in the DGG; otherwise, we define $similarity_a(n_1, n_2) = 0$.

We then define the ***similarity between two summaries s_1 and s_2***. Suppose s_1 and s_2 have *m* attributes a_1, …, a_m and s_1 is at level (n_{11}, …., n_{1m}) and s_2 is at level (n_{21}, …., n_{2m}), where n_{ij} denotes the concept level for the *i*th summary and the *j*th attribute. The similarity between s_1 and s_2 is defined as $similarity(s_1, s_2) = \frac{1}{m}\sum_{i=1}^{m} similarity_{a_i}(n_{1i}, n_{2i})$. We define the ***similar summaries of summary s*** as the set the summaries whose similarity values with *s* are greater than or equal to a threshold, i.e., $sim(s) = \{s' \mid similarity(s, s') \ge t\}$, where $0 \le t \le 1$ is a threshold.

For the weather data set, we set $\alpha = 0.8$ and $t = 0.95$. The analysis method is that for each iteration, we find the similar summaries *S* of the top summary and their average rankings. Then we accept the observed probability distribution of the top summary as the new estimates and propagate them to all the other nodes to start the next iteration. We then calculate the average rankings of the summaries in *S* in this iteration. Table 7.7 compares the average rankings of the similar summaries of the top summaries for the current and next iterations for the first 10 iterations. In all 10 iterations, the average rankings decrease significantly after propagation, which means that the similar summaries of the top summaries become less interesting after the user accepts the observed probability distributions of the top summaries.

Table 7.7 Average rankings of the similar summaries of the top summary

Iteration \ Average ranking	Current iteration	Next iteration
1	9.7	29.8
2	18.1	53.7
3	15.3	65.6
4	16.6	42.2
5	35.0	79.6
6	10.0	77.2
7	36.8	107.4
8	54.5	101.7
9	43.2	86.5
10	43.7	91.2

Based on the results of each iteration, we also observe that the user's estimates approach the real distributions as the process continues. Figure 7.3 shows the logarithm of the variances for the top summaries and top ten summaries in the fist sixteen iterations. Both the variances decrease as the number of the iterations increase in this example.

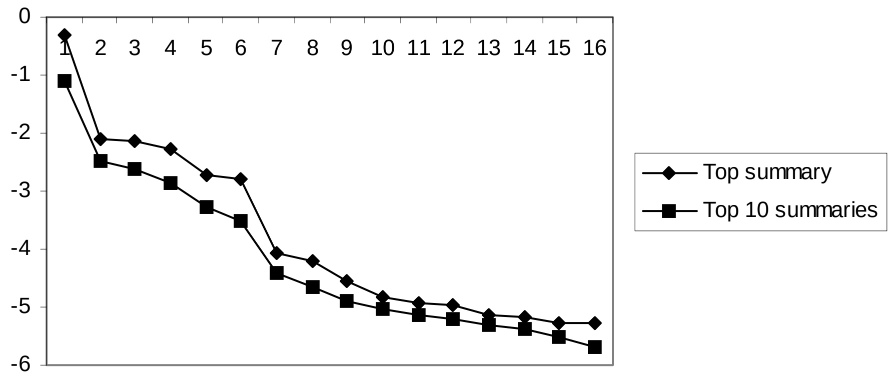


Figure 7.3 Logarithm of variances for the top and top 10 summaries in the first 16 iterations

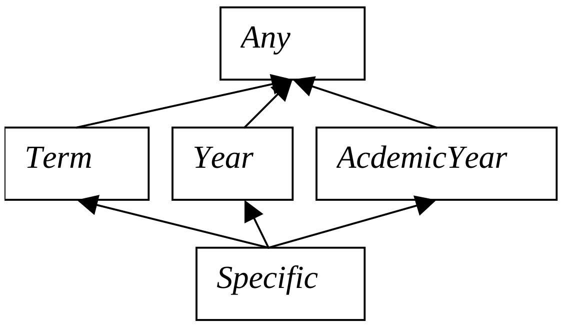
Lastly, we observe that although we originally expected to learn about the weather, as the knowledge discovery process continues, other relationships continue to be found, such as summer readings were taken more frequently than winter ones.

7.2 University of Regina Student Data Set

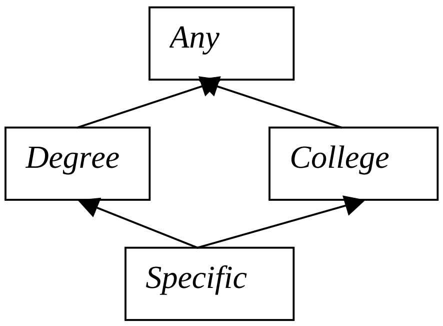
The objective of the application of the GSSM method to the University of Regina student data set is to find surprising knowledge about the distribution of students in terms of nationality, term, program, and GPA. We will also explore the correlation between these factors.

The University of Regina student data set contains student information from 1997 to 1999. The attributes in the table include *student number*, *term code*, *hours*, *GPA*, *campus*, *college*, *program*, *major*, *city*, etc. There are 59,635 records in the table. We choose four attributes, *term code*, *progra*m, *city*, and *GPA*, for our experiments. The DGGs for these attributes are shown in Figure 7.4.

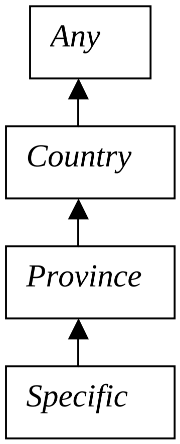
Attribute *Term* is generalized to *Term* (regardless of year), *Year* (starting from the winter term), and *Academic Year* (starting from the fall term). Attribute *Program* is generalized to *Degree* and *College*. Attribute City is generalized to *Province* and *Country*. Attribute *GPA* is generalized to *10-mark ranges* or an *ABCDFO scale*, corresponding to the marks of *A*, *B*, *C*, *D*, *F* (fail), and *Other*. The *10-mark ranges* and *ABCDFO* scale are further generalized to *PassOrNot*. In the final GenSpace graph, there are 400 nodes.



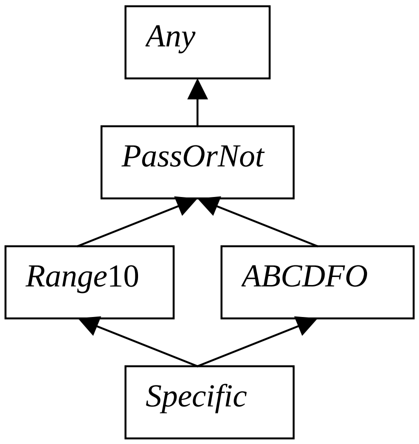
(a) DGG for attribute *Term*



(b) DGG for attribute *Program*



(c) DGG for attribute *City*



(d) DGG for attribute *GPA*

Figure 7.4 DGGs for attributes *Term*, *Program*, *City*, and *GPA*

7.2.1 Data Cleaning

The student data set has missing values and noises. Before we begin the mining process, we need to clean the data set.

1. Missing values.

A value in a certain conceptual level for an attribute may be missing in the original table. In some cases, this is caused by the inner logic of the data. For example, we have *Singapore* at the *City* level and *Country* level. But there is no corresponding value at the *Province* level. In this case, we simply add *Singapore* at the province level.

In other cases, missing values are caused by the ignorance of the operator. For example, we have the tuple "*Regina*, _, *Canada*" where the value for the *Province* level is missing. If all the values in different levels are missing, we replace the empty entry with "*other*". Therefore, we replace "_, _, _" with "*other*, *other*, *other*". If in a record, a value is missing, we first check if it appears in another record in the same context. If so, we use that value to replace "_", otherwise, we concatenate "*other*" and the concept name to replace "_". For example, in some records, we have "*Regina*, _, *Canada*". However, we can find records with "*Regina*, *SK*, *Canada*". So we replace "_" with "*SK*". In some records, we have "*Nottingham*, _, *UK*", and we could not find a name at the *Province* level corresponding to *Nottingham*, *UK* in the data set. Therefore, we replace it with "*Nottingham*, *OtherUKProvince*, *UK*". Similarly, we replace "_, *SK*, *Canada*" with "*OtherSKCity*, *SK*, *Canada*".

2. Spelling errors

There are some spelling errors in the student data set. Some of these errors will influence the mining results. For example, the city "*Regna*" is supposed to be "*Regina*". This kind of error will change the output, but it will not be an obstacle for measuring the performance of our algorithms. Other errors will make the generalization relations inconsistent and prevent the application of the algorithms. For example, the error in "*Regina*, *SJ*" makes the system confused whether to generalize *Regina* to *SK* or *SJ*. We consider this kind of error as inconsistent data error and discuss it in the next section.

3. Inconsistent data

To make the DGGs consistent, we need to guarantee the one-to-many relations between any pair of child-parent nodes. However, in the original table, we have inconsistencies because of misspelling errors, the multiple forms of the same name, or the inner logic of the data. For example, we have "*Regina, SK*" and "*Regina, Saskatchewan*" in the table. We have to convert one of them to the other to make sure that *Regina* will be mapped to a single province name without ambiguity. In another case, we have "*Lloydminster*, *AB*" and "*Lloydminster, SK*". This violation of one-to-many relation is because one part of the city belongs to *Saskatchwan*, and the other belongs to *Alberta*. In this case, we convert "*Lloydminster*, *AB*" to "*LloydminsterAB*, *AB*", and "*Lloydminster*, *SK*" to "*LloydminsterSK*, *SK*", i.e., consider the city as two cities. This conversion restores the consistency of the DGGs.

Our cleaning process does not aim to obtain 100% accurate and complete data. Instead, our objective is to make the data sufficiently consistent that our algorithm can be applied without losing existing information.

7.2.2 Mining Process

Initially, we assume an even distribution for students among all the countries present in the dataset. We assume that the distribution for the students in each college is uniform. We assume an even distribution for the students in the winter, summer and fall terms. We also assume that 90% of the students pass the classes they took. Finally, we assume that the four attributes are independent.

After the first iteration, we obtain the top ten summaries as shown in Table 7.8. From this table we can see that the summary "*Nation*, *Any*, *Any*, *Any*" is the most interesting one. Because we initially assumed uniform distribution among countries, we can see that the observed distribution is far from our estimates, i.e., it is far from a uniform distribution. To confirm this knowledge, we look into this summary and find 94.1% of students are from Canada. Now we know that from 1997 to 1999, most of the students attending the University of Regina are from Canada. We accept this observed distribution and propagate it throughout the GenSpace graph and start the second iteration.

Table 7.8 Top 10 summaries after the first iteration

City	**Program**	**Term**	**GPA**	**Variance**
Nation	Any	Any	Any	3.39e-2
Any	Any	Term	Any	2.84e-2
Any	Any	Academicyear	Any	2.78e-2
Nation	Any	Any	Passornot	2.40e-2
Any	Any	Year	Any	1.08e-2
Any	Any	Academicyear	Passornot	1.01e-2
Any	Any	Term	Passornot	1.00e-2
Prov	Any	Any	Any	7.71e-3
Nation	Any	Academicyear	Any	6.24e-3
Prov	Any	Any	Passornot	5.97e-3

After the second iteration, we obtain the top ten summaries in Table 7.9. Comparing this result with that of the first iteration, we can see that the summary "*Nation*, *Any*, *Any*, *An*y" has disappeared. The summaries involving *Nation* (the fourth and the ninth summaries) have disappeared from the top 10 list as well. This is due to updating the user's estimates concerning *Nation*, which leads to previously interesting summaries being regarded as uninteresting. As a result, the second most interesting summary, "*Any*, *Any*, *Term, Any*" is now on top. When we look at this summary, we find that the distribution of the students attending for the terms (*Winter*, *Spring/Summer*, *Fall*) is [0.39, 0.14, 0.47], which means that significantly fewer students take classes in the Spring/Summer term.

Table 7.9 Top 10 summaries after the second iteration

City	Program	Term	GPA	Variance
Any	Any	Term	Any	2.83e-2
Any	Any	Academicyear	Any	1.08e-2
Any	Any	Term	Passornot	9.98e-3
Any	College	Any	Any	4.58e-3
Any	Any	Academicyear	Passornot	4.25e-3
Any	Any	Any	Passornot	3.59e-3
Any	Any	Any	ABCDFO	3.19e-3
Any	Any	Any	Range10	2.05e-3
Any	College	Any	Passornot	1.92e-3
Any	Any	Year	Any	1.82e-3

Having updated and propagated the estimates for the *Term*, we obtain the results shown in Table 7.10 after the third iteration. In this table, the top three summaries for the second iteration have disappeared. Two of these summaries involve attribute *Term* at the *Term* level. It is reasonable to our estimation since we adjusted the estimates for *Term*. Checking the summay "*Any*, *College*, *Any*, *Any*", we obtain the following rules.

```
More students (27.39%) have college = AR than estimated (7.72%).
Fewer students (2.12%) have college = SP than estimated (7.68%).
Fewer students (2.30%) have college= PA than estimated (7.66%).
Fewer students (3.91%) have college = FA than estimated (7.72%)
More students (11.28%) have college = SC than estimated (7.70%)
Fewer students (4.78%) have college= EX than estimated (7.68%)
```

```
More readings (10.32%) have college = ED than estimated
(7.66%)
Fewer readings (6.16%) have college= EN than estimated
(7.68%)
More readings (9.05%) have college = SW than estimated
(7.68%)
```

Table 7.10 Top 10 summaries after the third iteration

City	Program	Term	GPA	Variance
Any	College	Any	Any	4.58e-3
Any	Any	Any	Passornot	3.66e-3
Any	Any	Any	ABCDFO	3.26e-3
Any	Any	Any	Range10	2.13e-3
Any	College	Any	Passornot	1.92e-3
Nation	College	Any	Any	7.92e-4
Any	College	Academicyear	Any	6.61e-4
Any	College	Term	Any	6.17e-4
Any	College	Year	Any	5.54e-4
Nation	College	Any	Passornot	4.69e-4

From the results from the fourth iteration (Table 7.11), the following rule is produced for the most interesting summary "*Any*, *Any*, *Any*, *Passornot*":

```
Fewer students (5.64%) failed classes than estimated (9.93%)
```

Table 7.11 Top 10 summaries after the fourth iteration

City	Program	Term	GPA	Variance
Any	Any	Any	Passornot	3.68e-3
Any	Any	Any	ABCDFO	3.14e-3
Any	Any	Any	Range10	2.11e-3
Any	Any	Year	ABCDFO	4.12e-4
Any	Any	Academicyear	ABCDFO	3.63e-4
Any	Any	Term	ABCDFO	3.62e-4
Any	Any	Year	Passornot	3.30e-4
Any	Any	Year	Range10	3.16e-4
Any	Any	Academicyear	Passornot	2.85e-4
Any	Any	Term	Range10	2.63e-4

After the eleventh iteration, the top summary involves two attributes, College and Term (Table 7.12). In the summary, some of the rules are presented as follows.

```
Fewer students (2.65%) have college = AR and term = 2 than
estimated (3.95%)
More students (13.58%) have college = AR and term = 3 than
estimated (12.55%)
More readings (2.14%) have college = GS and term = 2 than
estimated (1.30%)
More readings (3.50%) have college = EP and term = 3 than
estimated (2.72%)
```

Initially we assumed that the attributes Program and Term were independent. From the discovered rules, we can see that students of college of *AR* are less likely to attend classes in *Spring/Summer* and more likely in *Fall*. Graduate students are more likely to attend the classes in *Sping/Summer* than estimated.

Now we revise our estimates based on the observations of this summary.

Table 7.12 Top 10 summaries after the eleventh iteration

City	**Program**	**Term**	**GPA**	**Variance**
Any	College	Term	Any	1.87e-5
Any	coll	Any	Any	1.63e-5
Any	Any	Term	ABCDFO	1.53e-5
Any	Any	Term	Any	1.25e-5
Any	Any	Term	Range10	1.15e-5
Any	College	Any	Passornot	1.07e-5
Any	College	Academicyear	Any	1.05e-5
Any	Any	Term	Passornot	7.39e-6
Any	College	Any	Grade4	7.29e-6
Any	College	Term	Passornot	7.14e-6

After the twelfth iteration, the top summary is “Any, College, Any, Passornot”, as shown in Table 7.13, which means that there is a correlation between Program and GPA based on the user’s estimates. The top three rules produced on this summary are:

```
More students (1.50%) have college = EP and passornot = fail
than estimated (0.38%)
Fewer students (25.21%) have college = AR and passornot =
pass than estimated (25.76%)
Fewer students (0.04%) have college = GS and passornot =
fail than estimated (0.43%)
```

Table 7.13 Top 10 summaries after the twelfth iteration

City	Program	Term	GPA	Variance
Any	College	Any	Passornot	1.63e-5
Any	Any	Term	Passornot	1.19e-5
Any	Any	Academicyear	Passornot	1.18e-5
Any	Any	Year	Any	1.11e-5
Any	Any	Term	ABCDFO	1.06e-5
Any	Any	Year	Passornot	9.74e-6
Any	Any	Academicyear	Any	8.77e-6
Any	Any	Term	Range10	8.08e-6
Any	College	Any	ABCDFO	7.19e-6
Any	Degree	Any	Passornot	6.25e-6

Figure 7.5 shows the logarithms of the variances for the top and top 10 summaries in twenty iterations. Both measures are roughly a decreasing function to the number of iterations.

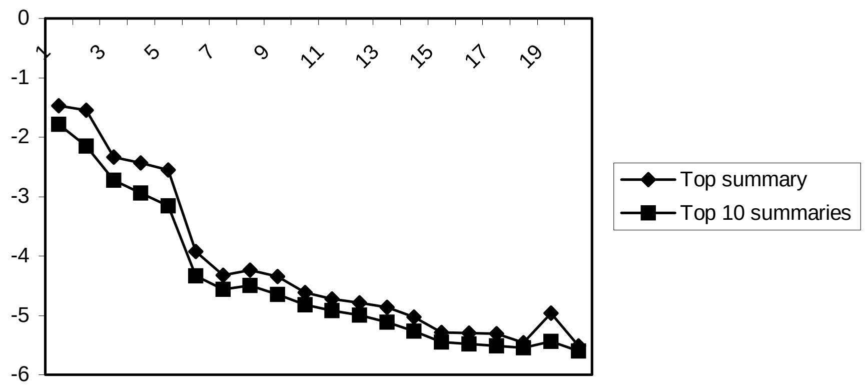


Figure 7.5 Logarithm of variances for the top and top 10 summaries for the first 20 iterations

We also experimented using the Schutz (relative variance) measure as an alternative interestingness measure. Since the Schutz measure does not have the bias towards the higher-level summaries as variance does, the top summary in each iteration is more complex than that when variance is used as interestingness measure. Table 7.14 shows the top summaries for the first 10 iterations with Schutz measure. Table 7.15 compares the size (number of records) of the top summaries mined with variance and Schutz measures for the first 10 iterations. The sizes of the top summaries mined with the Schutz measure are greater than those of top summaries mined with the variance measure. We also found that the Schutz measure decreases during the iterations as the variance does. Figure 7.6 shows the Schutz values for the top summaries in the first 10 iterations.

Table 7.14 Top summaries for the first 10 iterations with Schutz measure

Top summary / Iteration	City	Program	Term	GPA
1	Prov	Degree	Year	Grade4
2	Any	College	Term	Passornot
3	Prov	Any	Academicyear	Passornot
4	Prov	College	Term	Passornot
5	Any	Degree	Academicyear	Passornot
6	Nation	College	Academicyear	Range10
7	Prov	Degree	Term	Range10
8	Any	College	Year	Passornot
9	Prov	Degree	Acdemicyear	Range10
10	Nation	College	Year	Grade4

Table 7.15 Sizes of the top summaries mined with the variance and Schutz measures

Size / Iteration	Variance	Schutz measure
1	17	2126
2	3	76
3	13	191
4	2	538
5	6	278
6	3	621
7	64	2350
8	8	78
9	68	2187
10	6	568

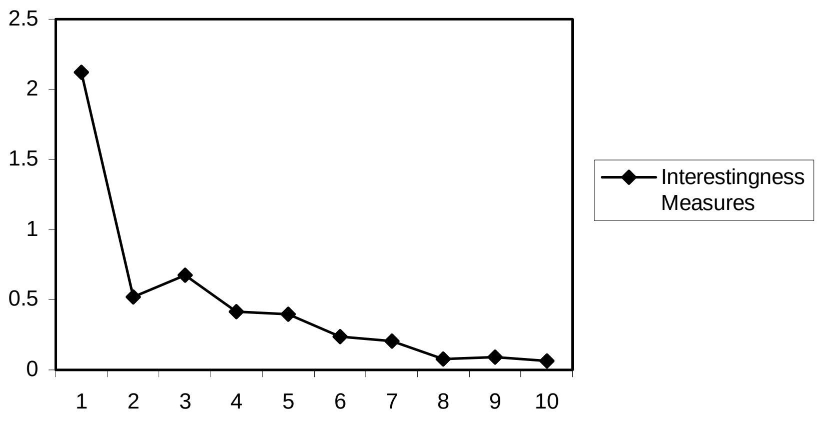
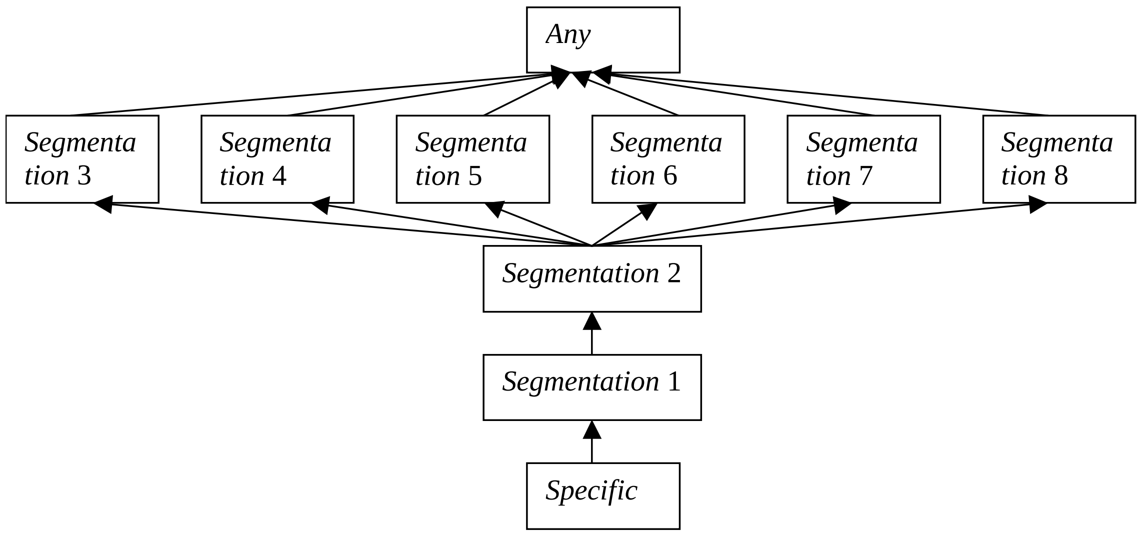


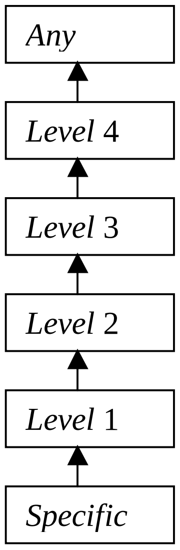
Figure 7.6 Schutz measure for the top summaries in the first 10 iterations

7.3 A Customer Data Set

The third application is conducted on a customer data set, which contains 1,154,449 tuples. We concentrated on the relationship between the type of business that customer is associated with (*CustomerID*) and the equipment or services that they requested (*ServiceID*). The DGGs for the two attributes are shown in Figure 7.7.



(a) DGG for attribute *CustomerID*



(b) DGG for attribute *ServiceID*

Figure 7.7 DGGs for the customer data set

Initially we assume that the distribution at the *Segmentation* 7 level for attribute *CustomerID* is even, i.e., the estimates for *competitive services*, *industrial services*, *institutions*, *others*, and *unknown* are all 0.2. We also assume that the distribution for *level* 4 of *serviceID* is even.

The top 6 summaries from the first iteration are listed in Table 7.16. Further checking the top summary *Segmentation* 6, we get the following rules.

```
Fewer readings (0.519382%) have Segmentation 6 = UNKNOWN
than estimated (15.3061%).
More readings (64.2468%) have Segmentation 6 = SERVICES than
estimated (33.6927%).
Fewer readings (2.13773%) have Segmentation 6 = OTHERS
EXISTING IN DATABASE than estimated (20.4082%).
```

Figure 7.8 shows the observed distribution for attribute *CustomerID* at the *Segmentation* 6 level.

Table 7.16 Top six summaries from the first iteration

CustomerID	**ServiceID**	**Variance**
Segmentation 6	Any	4.97e-2
Segmentation 3	Any	3.77e-2
Segmentation 4	Any	3.70e-2
Segmentation 7	Any	3.23e-2
Segmentation 8	Any	1.97e-2
Segmentation 5	Any	1.62e-2

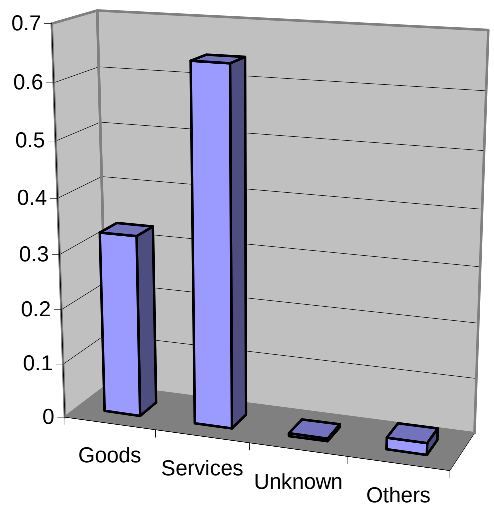


Figure 7.8 Distribution in summary *Segmentation* 6, *Any*

After the second iteration, we obtain the top summaries in Table 7.17 and the distribution for the top summary *Segmentation* 3, *Any* in Figure 7.9.

Table 7.17 Top six summaries from the second iteration

SIC	**USOC**	**Variance**
Segmentation 3	Any	1.63e-2
Any	Level 4	1.48e-2
Segmentation 4	Any	1.47e-2
Segmentation 5	Any	6.93e-3
Segmentation 8	Any	6.88e-3
Segmentation 7	Any	3.36e-3

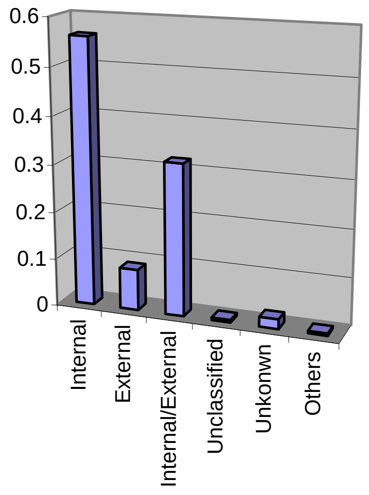


Figure 7.9 Distribution in summary *Segmentation* 3, *Any*

After the eighth iteration, we obtain the top summaries in Table 7.18. The top summary involves both attributes, *CustomerID* at level *Segmentation* 7 and *ServiceID* at *Level* 4. Checking the summary, we found that most transactions are with *competitive services*, *industrial services*, and *institutions*, regarding *centrex voice hub* and *single line telephones* (See Figure 7.10).

Table 7.18 Top summaries from the eighth iteration

SIC	**USOC**	**Variance**
Segmentation 7	Level 4	1.16e-4
Segmentation 6	Level 4	1.05e-4
Any	Level 3	7.91e-5
Segmentation 4	Level 4	6.93e-5
Segmentation 8	Level 4	6.37e-5
Segmentation 3	Level 4	6.33e-5

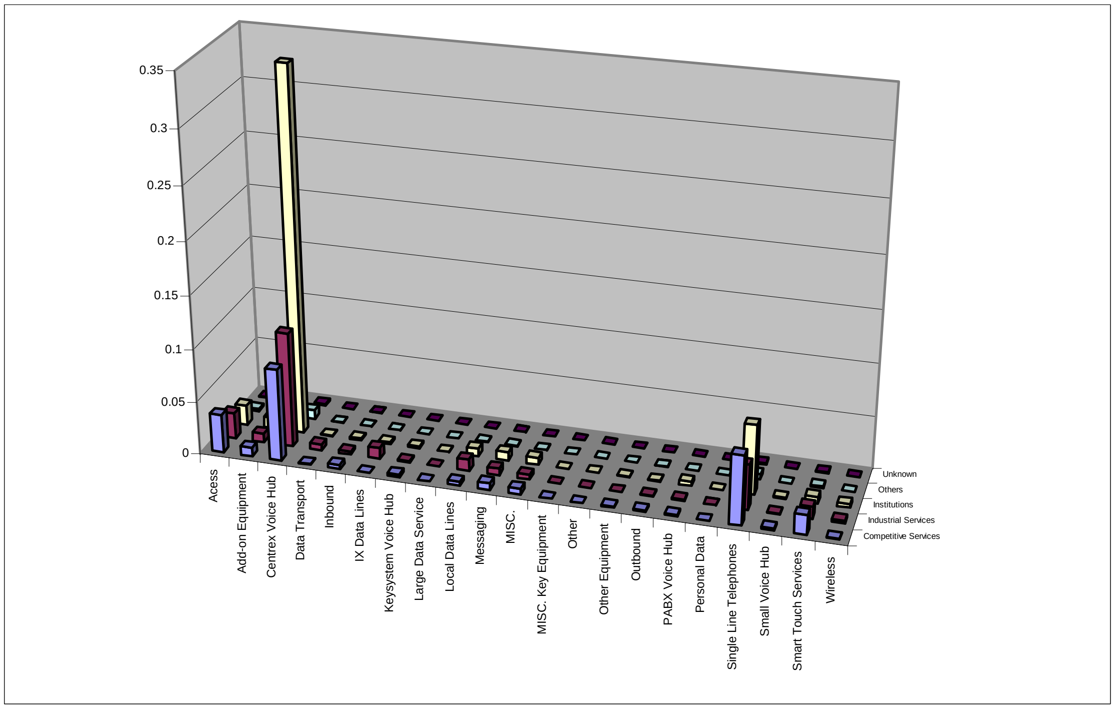


Figure 7.10 Distribution in summary *Segmentation* 7, *Level* 4

7.4 Summary

In this chapter, we demonstrated the effectiveness of the GSSM method on three real data sets, the Saskatchewan weather data set, the University of Regina student data set, and a customer data set. Based on the mining process and results, we have the following observations. First, the GSSM mining process integrates belief revision. The GSSM finds the most interesting evidence (summary) to present to the user. Based on the evidence, the user revises his/her belief, which is simulated by the GSEP process. Based on the revised belief, GSSS then finds the most interesting evidence in the next iteration. Experimental results show that during the mining process, the interestingness values of the top summaries, which represent the distance between the user's belief and the observed distribution in the data, decrease. This coincides with our estimation that the user’s knowledge becomes closer and closer to the distribution represented by the data during the mining process.

Secondly, the GSSM method can greatly facilitate the summary exploration process. The initial knowledge of the data can easily be specified by the user. In most of cases in the experiments, we assume an even probability distribution for each attribute at a certain level. After that, the process can be completely automated.

Thirdly, the GSSM method responds to changes in the user's knowledge during the knowledge discovery process. If a summary has been presented to the user as the most interesting summary, its closely related summaries will become less interesting in the next iteration. The example of Pay-TV shows in Chapter 1 can be well tackled by the GSSM method.

CHAPTER 8

CONCLUSIONS

In this thesis, we introduced the GenSpace summary mining (GSSM) problem and proposed a solution to this problem. The original contributions of the thesis include:

(1) A framework for summary mining, called GSSM, based on belief revision is proposed. In this framework, the system consists of two parts: GSEP for propagating user's estimates and GSSS for selecting the interesting summaries in the GenSpace graph.

(2) We proposed three GSEP principles and based on the principles, we formalized the GSEP problem as a linear constrained least square optimization problem. We also proposed a linear GSEP method as a heuristic method to the optimization based GSEP method with a more efficient calculation process.

(3) Two methods to find efficient propagation paths in GenSpace subgraphs are proposed. Experimental results show that these two methods can greatly reduce the propagation and storage costs, if some nodes at lower levels in the GenSpace graph or some *specific* nodes in ExGen graphs are defined as uninteresting nodes.

(4) Virtual bottom nodes are proposed to reduce the storage and time costs. Virtual bottom nodes can be applied together with the path selection algorithms to further improve the efficiency of the GSSM process.

(5) Interestingness measures that can incorporate a user's estimates are analyzed in the context of GSSM, especially in terms of pruning strategies.

The current version of the GSSM system has the following limitations.

We do not have an integrated user interface for the system. The user needs to edit the DGG files for each attribute manually. The system outputs the most interesting summaries in text files. For this system, a graphic user interface that can assist the users to define the input and view the output will be useful.

The GSSM method needs more calculation than the most other summarization algorithms for the following two reasons. First, other than the selected materialization method [Harinarayan et al., 1996], the well-known summarization algorithms do not evaluate multiple ways of aggregations. The selected materialization method aims to answer a query to a specific summary efficiently, i.e., it only needs to aggregate one summary at one time. Secondly, GSSM not only needs to aggregate the summaries, but also needs to propagate estimates and calculate the interestingness measures. Therefore, it needs more calculations than all other summarization methods. However, none of the existing aggregation methods can find interesting summaries based on the user's estimate as GSSM does. As well, none of the existing aggregation methods except the selected materialization method has multiple ways of aggregation as GSSM does.

In the future, many topics regarding the GSSM system deserve further research. First, an efficient I/O algorithm for GSSM during the mining process is desirable for large data sets. In our implementation, we load the entire bottom node in memory and only output the top summaries in each round of the GSSM process. If the bottom node is too large to be loaded in memory at the same time, an efficient I/O method is desirable. The Partitioned-Cube aggregation method has efficient I/O cost of $O(kr)$, where k is the number of attributes and r is the number of records. However, it can only be applied to the case without conceptual generalization. How to partition the bottom node is not an important issue in the Partitioned-Cube. However, in the GSSM process, the partition method will influence the I/O efficiency. The I/O problem in GSSM process is to find an appropriate partition method and choose propagation paths such that the I/O cost is minimum. In the future, we could design an algorithm to deal with this problem and integrate it into our GSSM system.

In our current implementation, only count is aggregated and used in the interestingness measures. In future research, other aggregation functions such as sum, mimimum, and maximum could be readily used. As well, more complex aggregation functions, such as average, could also be used. Suitable pruning strategies based on these functions might be found for the GSSM process.

Interestingness measure selection remains an open issue in the area of data mining. In our experiments, we only used the Schutz and variance measures. In the future, additional experiments with more interestingness measures could be performed to study their effects on the GSSM problem.

An estimate propagation and pruning strategy could be developed to mine other kinds of knowledge that is present in the GenSpace graphs, such as anomaly rules. For example, suppose that record *abc* is generalized to *abc′* in a GenSpace graph. We can find the distance of the observed and estimated percentage of *c* to *c′* in the context of *ab*.

The connection between the GSEP and Bayesian belief updating, especially how to use conditional independence among variables and Bayesian belief updating technique to improve the efficiency of the GSEP process, could be further studied.

When we use chi-square to find the interesting records in summaries, we set the level of significance at 0.05. However, with multiple tests in our case, the probability of false rejection can be highly inflated, which will result in Type I error. In the future, we will use a more stringent level of significance cutoff for the individual tests based on Bonferroni inequality.

Currently, we assume that the user accepts the observed probability distribution of the most interesting summary as the new estimates to facilitate the analysis of the effectiveness of GSSM method. In the future, experiments with human users would be useful. In particular, users should be allowed to vary the estimates between iterations of the GSSM process.

List of References

[Aha et al., 1991] Aha, D.W., Kibler, D., and Albert, M.K. Instance based learning algorithms. *Machine Learning*, 6, 37-66, 1991.

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. Fast Algorithms for mining association rules. *Proceedings of the 20th International Conference on Very Large Databases*, 487-499, Santiago, Chile, September 1994.

[Agrawal et al., 1993] Agrawal, R., Imielinski, T., and Swami, A. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 207-216, Washington, DC, 207-216, 1993.

[Agarwal et al., 1996] Agarwal, S., Agrawal R., Deshpande, P. M., Gupta A., Naughton, J. F., Ramakrishnan, R., and Sarawagi, S. On the computation of multidimensional aggregates. *Proceedings of the 22nd VLDB Conference*, 506-521, Bombay, India, 1996.

[Agrawal et al., 1998] Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Record*, 27(2), 94-105, 1998.

[Allison, 1978] Allison, P.D. Measures of inequality. *American Sociological Review*, 43, 865-880, 1978.

[Atkinson, 1970] Atkinson, A.B. On the measurement of inequality. *Journal of Economic Theory*, 2, 244-263, 1970.

[Attaran and Zwick, 1989] Attaran, M. and Zwick, M. An information theory approach to measuring industrial diversification. *Journal of Economic Studies*, 16, 19-30, 1989.

[Bargiela and Pedrycz, 2002] Bargiela, A. and Pedrycz, W. *Granular Computing: An Introduction*. Kluwer Academic Publishers, Boston, 2002.

[Bastide et al., 2000] Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., and Lakhal, L. Mining minimal non-redundant association rules using frequent closed itemsets. *Proceedings of the First International Conference on Computational Logic*, 972-986, London, UK, July 2000.

[Bay and Pazzani, 1999] Bay, S.D. and Pazzani, M.J. Detecting change in categorical data: mining contrast sets. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining* (*KDD99*), 302-306, San Diego, USA, 1999.

[Bay and Pazzani, 2001] Bay, S.D., and Pazzani, M.J. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5, 213-246, 2001.

[Bayardo, 1998] Bayardo, R.J. Efficiently mining long patterns from databases. *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (*SIGMOD'98*), 85-93, Seattle, WA, 1998.

[Beyer and Ramakrishnan, 1999] Beyer, K. and Ramakrishnan, R. Bottom-up computation of sparse and iceberg CUBEs. *Proceedings ACM SIGMOD International Conference on Management of Data* (*SIGMOD'99*), 359-370, Philadelphia, USA, 1999.

[Bishop, 1995] Bishop, C.M. *Neural Networks for Pattern Recognition*. Oxford University Press Publish, 1995.

[Brin et al., 1997] Brin, S., Motwani, R., and Silverstein, C. Beyond market basket: Generalizing association rules to correlations. *Proceedings of ACM SIGMOD International Conference on Management of Data* (*SIGMOD'97*), 265-276, Tucson, Arizona, 1997.

[Bulla, 1994] Bulla, L. An index of evenness and its associated diversity measure. *Oikos*, 70(1), 167-171, 1994.

[Cai et al., 1998] Cai, C.H., Fu, A.W., Cheng, C.H., and Kwong, W.W. Mining association rules with weighted items. *Proceedings of the International Database Engineering and Applications Symposium* (*IDEAS'98*), 68-77, Cardiff, Wales, UK, July 1998.

[Chan et al., 2003] Chan, R., Yang, Q., and Shen, Y. Mining high utility itemsets. *Proceedings of the 2003 IEEE International Conference on Data Mining* (*ICDM2003*), 19-26, Melbourne, FL, 2003.

[Chaudhuri and Dayal, 1997] Chaudhuri, S., and Dayal, U. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26, 65-74, 1997.

[Carvalho and Freitas, 2000] Carvalho, D.R. and Freitas, A.A. A genetic algorithm-based solution for the problem of small disjuncts. *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery* (*PKDD2000*), 345-352, Lyon France, 2000.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. The CN2 Induction Algorithm. *Machine Learning*, 2, 261-283, 1989

[Delgrande et al., 2004] Delgrande, J.P., Nayak, A.C., and Pagnucco, M. Gricean belief change. *Studia Logica*, 68, 1-18, 2004.

[Dong and Li, 1998] Dong G. and Li, J. Interestingness of discovered association rules in terms of neighborhood-based unexpectedness. *Proceedings of Second Pacific Asia Conference on Knowledge Discovery in Databases* (*PAKDD98*), 72-86, Melbourne, 1998.

[Ecoregions] http://interactive.usask.ca/ski/environment/ecoregions/index.html.

[Fabris and Freitas, 2001] Fabris, C.C. and Freitas, A.A Incorporating deviation-detection functionality into the OLAP paradigm. *Proceedings of XVI Brazilian Symposium on Databases* (*SBBD'2001*), 274-285, Rio de Janeiro, Brazil, October 2001.

[Fayyad et al., 1996] Fayyad, U.M., Piatetsky-Shapiro, G., and Smyth, P. From data mining to knowledge discovery: an overview. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, R., and Uthurusamy, R. (eds), *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1-34, 1996.

[Fletcher, 1987] Fletcher, R. *Practical Methods of Optimization*, John Wiley and Sons, 1987.

[Geng and Hamilton, 2002] Geng, L. and Hamilton, H.J. ESRS: A case selection algorithm using extended similarity-based rough sets. *Proceedings of the Second IEEE International Conference on Data Mining* (*ICDM'02*), 609-612, Maebashi, Japan, December, 2002.

[Geng and Hamilton, 2003a] Geng, L. and Hamilton, H.J. Automated case generation from databases using similarity-based rough approximation. In Abraham, A., Jain, L., and Kacprzyk, J. (eds.), *Recent Advances in Intelligent Paradigms and Application*s, Springer Verlag, Berlin, March, 2003. ISBN 3790815381.

[Geng and Hamilton, 2003b] Geng, L and Hamilton, H.J. *Estimate Propagation in ExGen Graphs for Summarization*. Tech. Report CS-2003-03, Department of Computer Science, University of Regina, May 2003.

[Geng and Hamilton, 2004] Geng, L., and Hamilton, H.J. Finding interesting summaries in GenSpace Graphs efficiently. *Proceeding of Canadian Conference on AI 2004*, 89-104, London, Canada, May 2004.

[Gray and Orlowska, 1998] Gray, B. and Orlowska, M.E. CCAIIA: Clustering categorical attributes into interesting association rules. *Proceedings of the Second Pacific Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD98*), 132-143, Melbourne, 1998.

[Grzymala-Busse, 1992] Grzymala-Busse, J. W. LERS – A system for learning from examples based on rough sets. In Slowinski, R. (ed.), *Intelligent Decision Support: Handbook of Applications and Advances of Rough Sets Theory*, Kluwer Academic Publishers, 3-18, 1992.

[Guha et al., 1998] Guha, S., Rastogi, R., and Shim, K. CURE: An efficient clustering algorithm for large databases, *SIGMOD Record*, 27(2), 73-84, 1998.

[Hamilton et al., 2003] Hamilton, H.J., Geng, L., Findlater, L, and Randall, D.J. Spatio-temporal data mining with expected distribution domain generalization graphs. *Proceedings of the 10th Symposium on Temporal Representation and Reasoning / International Conference on Temporal Logic (TIME-ICTL 2003)*, IEEE CS Press, Cairns, 181-191, Australia, July 2003.

[Hamilton et al., 2005] Hamilton, H., Geng, L., Findlater, L., and Randall D. Efficient spatio-temporal data mining with GenSpace Graphs. *Accepted by the Journal of Applied Logic Special Issue.*

[Han, 1998] Han, J. Towards on-line analytical mining in large databases. ACM SIGMOD Record, 27, 97-107, 1998.

[Han et al., 2001] Han, J., Pei, J., Dong, G., and Wang, K. Efficient computation of iceberg cubes with complex measures. *Proceedings of the 2001 ACM SIGMOD International Conference on Managemnet of Data*, 1-12, Santa Barbara, California, 2001.

[Han et al., 2004] Han, J., Pei, J., Yin Y., and Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1), 53-87, 2004.

[Harinarayan et al., 1996] Harinarayan, V., Rajaraman, A., and Ullman, J. D. Implementing data cubes efficiently. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 205-216, Montreal, 1996.

[Hart, 1971] Hart, P.E. Entropy and other measures of concentration. *Journal of the Royal Statistical Society, Series A*, 134, 73-85, 1971.

[Herzig and Rifi, 1999] Herzig, A. and Rifi, A. Propositional belief base update and minimal change. *Artificial Intelligence*, 115(1), 107-138, 1999.

[Hilderman et al., 1998] Hilderman, R.J., Carter, C.L., Hamilton, H.J., and Cercone, N. Mining market basket data using share measures and characterized itemsets. *Proceedings of the Second Pacific Asia Conference on Knowledge Discovery in Databases* (*PAKDD98*), 72-86, Melbourne, 1998.

[Hilderman and Hamilton, 2001] Hilderman, R. J. and Hamilton, H., J. *Knowledge Discovery and Measures of Interest*, Kluwer Academic Publishers, 2001.

[Jensen, 1996] Jensen, F.V. *An Introduction to Bayesian Networks*, Springer, New York, 1996.

[Katsuno and Mendelzon, 1991] Katsuno, H. and Mendelzon, A. O. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52, 253-294, 1991.

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P.J. *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, 1990.

[Klosgen, 1996] Klosgen, W. Explora: A multipattern and multistrategy discovery assistant. *Advances in Knowledge Discovery and Data Mining* (Fayyard et al. eds), AAAI Press/MIT Press, California, 249-271, 1996.

[Knorr et al., 2000] Knorr, E.M., Ng, R.T., and Tucakov, V. Distance based outliers: Algorithms and applications, *International Journal on Very Large Databases*, 8, 237-253, 2000.

[Jensen, 1996] Jensen, F. V. *An introduction to Bayesian networks*, Springer Publishers, New York, 1996.

[Lavrac et al., 1999] Lavrac, N., Flach, P., and Zupan, B. Rule evaluation measures: A unifying view. *Proceedings of the Ninth International Workshop on Inductive Logic Programming* (*ILP'99*), 174-185, Bled, Slovenia, June 1999.

[Lenca et al., 2004] Lenca P., Meyer P., Vaillant B., Lallich S. *A Multicriteria Decision Aid for Interestingness Measure Selection.* Technical Report LUSSI-TR-2004-01-EN, May 2004, LUSSI Department, GET / ENST Bretagne.

[Lewontin, 1972] Lewontin, R.C. The apportionment of human diversity. *Evolutionary Biology*, 6, 381-398, 1972.

[Li and Hamilton, 2004] Li, G. and Hamilton, H.J. Basic association rules. *Proceedings of 2004 SIAM International Conference on Data Mining* (*SIAMDM04*), 166-177, Orlando, USA, April, 2004.

[Ling et al., 2002] Ling C., Chen, T., Yang Q., and Chen J. Mining optimal actions for profitable CRM. *Proceedings of the 2002 IEEE International Conference on Data Mining* (*ICDM2002*), 767-770, Maebashi City, Japan, 2002.

[Liu et al., 1997] Liu, B., Hsu, W., and Chen, S. Using general impressions to analyze discovered classification rules. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, 31-36, Newport Beach, California, USA, August 1997.

[Liu et al., 1999] Liu, B., Hsu, W., Mun L., and Lee, H. Finding interesting patterns using User estimates. *IEEE Transactions on Knowledge and Data Engineering*, 11(6), 817-832, 1999.

[Lu et al., 2001] Lu, S., Hu, H., and Li, F. Mining weighted association rules. *Intelligent Data Analysis*, 5(3), 211-225, 2001.

[Makinson, 1993] Makinson, D. Five faces of minimality. *Studia Logica*, 52, 339-379, 1993.

[Molinari, 1989] Molinari, J. A calibrated index for the measurement of evenness. *Oikos*, 56(3), 319-326, 1989.

[Ohsaki et al., 2004] Ohsaki, M., Kitaguchi, S., Okamoto, K., Yokoi, H., and Yamaguchi, T. Evaluation of rule interestingness measures with a clinical dataset on hepatitis.

Proceedings of the Eighth European Conference on Principles of Data Mining and Knowledge Discovery (*PKDD2004*), 362-373, Pisa, Italy, 2004.

[Padmanabhan and Tuzhilin, 1998] Padmanabhan, B. and Tuzhilin, A. A belief-driven method for discovering unexpected patterns. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (*KDD98*), 94-100, New York City, 1998.

[Padmanabhan and Tuzhilin, 2000] Padmanabhan, B. and Tuzhilin, A. Small is beautiful: Discovering the minimal set of unexpected patterns. *Proceedings of Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining* (*KDD2000*), 54-63, Boston, USA, 2000.

[Peet, 1974] Peet, R.K. The measurement of species diversity. *Annual Review of Ecology and Systematics*, 5, 285-307, 1974.

[Piatesky-Shapiro and Matheus, 1994] Piatesky-Shapiro, G. and Matheus, C. The interestingness of deviations. *Proceedings of KDD Workshop 1994* (*KDD94*), 77-87, Seattle, USA, 1994.

[Piatetsky-Shapiro, 1991] Piatetsky-Shapiro, G. Discovery, analysis and presentation of strong rules. *Knowledge Discovery in Databases* (Piatetsky-Shapiro and Frawley eds), 229-248. MIT Press, Cambridge, MA, 1991.

[Papadimitriou and Steiglitz, 1982] Papadimitriou, C.H. and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*, chapter 11, 247-254. Enlewood Cliffs, N.J., Prentice Hall, 1982.

[Pawlak, 1992] Pawlak, Z. *Rough Sets: Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, Norwell, MA, USA , 1992.

[Quinlan, 1993] Quinlan, J.R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Randall et al., 1999] Randall, D.J., Hamilton, H.J., and Hilderman, R.J. Temporal generalization with domain generalization graphs. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(2), 195-217, 1999.

[Ross and Srivastava, 1997] Ross, K. A., and Srivastava, D. Fast computation of sparse datacubes. *Proceedings of the 23rd International Conference on Very Large Databases*, 116-125, Athens, Greece, 1997.

[Rott, 2000] Rott, H. Two dogmas of belief revision. *Journal of Philosophy*, 97, 503-522, 2002.

[Sahar, 1999] Sahar, S. Interestingness via what is not interesting. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 332-336, San Diego, USA, August 1999.

[Sarawagi, 1999] Sarawagi, S. Explaining differences in multidimensional aggregates. *Proceedings of the 25th International Conference on Very Large Databases* (*VLDB*), 42-53, Edinburgh, Scotland, 1999.

[Sarawagi et al., 1996] Sarawagi, S., Agrawal, R., and Gupta, A. *On Computing the Data Cube*. Research Report RJ10026, IBM Almaden Research Center, 1996.

[Sarawagi et al., 1998] Sarawagi, S., Agrawal, R., and Megiddo, N. Discovery driven exploration of OLAP data cubes. *Proceedings of the Sixth International Conference of Extending Database Technology* (*EDBT'98*), 168-182, Valencia, Spain, March 1998.

[Schittkowski, 1985] Schittkowski, K. NLQPL: A FORTRAN-subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5, 485-500, 1985.

[Schutz, 1951] Schutz, R.R. On the measurement of income inequality. *American Economic Review*, 41, 107-122, March 1951.

[Shen et al., 2002] Shen, Y. D., Zhang, Z. and Yang, Q. Objective-oriented utility-based association mining. *The Second IEEE International Conference on Data Mining*, 426-433, Maebashi City, Japan, December 2002.

[Silvescu et al., 2001] Silvescu A., Reinoso-Castillo, J, and Honavar V. Ontology-driven information extraction and knowledge acquisition from heterogeneous, distributed, autonomous biological data. *Proceedings of the IJCAI-2001 Workshop on Knowledge Discovery from Heterogeneous, Distributed, Autonomous, Dynamic Data and Knowledge Sources*, 2001.

[Silberschatz and Tuzhilin, 1995] Silberschatz, A. and Tuzhilin, A. On subjective measures of interestingness in knowledge discovery. *First International Conference on Knowledge Discovery and Data Mining*, 275-281, Montreal, Canada, August 1995.

[Silberschatz and Tuzhilin, 1996] Silberschatz, A. and Tuzhilin, A. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering* 8(6), 970-974, 1996.

[Skowron and Stepaniuk, 2001] Skowron, A. and Stepaniuk, J. Information granules: towards foundations of granular computing. *International Journal of Intelligent Systems*, 16, 57-85, 2001.

[Smith and Liebman, 1980] Smith, J. M., and Liebman, J. S. An O(n^2) heuristic algorithm for the directed Steiner minimal tree problem. *Applied Mathematic Modeling*, 4(5), 369-375, 1980.

[Smyth and McKenna, 1999] Smyth, B. and McKenna, E. Building compact competent case-bases. *Proceedings of the Third International Conference on Case-based Reasoning*, 329-342, Munich, Germany, 1999.

[Tan et al., 2000] Tan, P and Kumar, V. *Interestingness Measures for Association Patterns: A Perspective*, Technical Report 00-036, Department of Computer Science, University of Minnesota, 2000.

[Tan et al., 2002] Tan, P., Kumar, V., and Srivastava, J. Selecting the right interestingness measure for association patterns. *Proceedings of Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining* (*KDD02*), 32-41, Edmonton, Canada, 2002.

[Val and Shoham, 1994] Val, A. D. and Shoham, Y. A unified view of belief revision and update. *Journal of Logic and Computation*, 4(5), 797-810, 1994.

[Wang et al., 2002] Wang, K., Zhou, S. and Han, J. Profit mining: from patterns to actions. *Proceedings of the Eighth Conference on Extending Database Technology* (*EDBT 2002*), 70-87, Prague, 2002.

[Webb et al., 2003] Webb, G. I., Butler, S., and Newlands D. On detecting differences between groups. *Proceedings of the Nineth ACM SIGKDD Internation Conference on Knowledge Discovery and Data Mining* (*KDD03*), 256-265, Washington DC, USA, 2003.

[Webb and Brain, 2002] Webb, G.I., and Brain, D. Generality is predictive of prediction accuracy. *Proceedings of the 2002 Pacific Rim Knowledge Acquisition Workshop* (*PKAW 2002*), 117-130, Tokyo, Japan, 2002.

[Wilson and Martines, 2000] Wilson, D.R. and Martines, T.R. An integrated instance-based learning algorithm. *Computational Intelligence*, 16(1), 1-28, 2000.

[Xiang, 2002] Xiang Y. *Probabilistic Reasoning in Multi-Agent Systems: A Graphical Models Approach*, Cambridge University Press, New York, 2002.

[Yao et al., 1999] Yao, Y.Y. and Zhong, N. An analysis of quantitative measures associated with rules. *Proceedings of the Third Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 479-488, Beijing, China, April 1999.

[Yao, 2000] Yao, Y.Y. Granular computing: basic issues and possible solutions. *Proceedings of the 5th Joint Conference on Information Sciences*, Volume I, 186-189, Atlantic City, New Jersey, USA, 2000.

[Yao et al., 2004] Yao, H., Hamilton, H.J., and Butz, C.J. A foundational approach for mining itemset utilities from databases. *Proceedings of 2004 SIAM International Conference on Data Mining* (*SDM04*), 482-486, Orlando, FL, April 2004.

[Zhang et al., 1996] Zhang, T., Ramakrishna, R., and Livny, R. BIRCH: An efficient data clustering method for very large databases. *SIGMOD Record*, 25(2), 103-114, 1996.

[Zhang et al., 2002] Zhang J., Silvescu A., and Honavar V. Ontology-driven induction of decision trees at multiple levels of abstraction. *Proceedings of Symposium on Abstraction, Reformulation, and Approximation*, Kananaskis, Canada, 2002.

[Zhang et al., 2004] Zhang, H., Padmanabhan, B., and Tuzhilin, A. On the discovery of significant statistical quantitative rules. *Proceedings of the Tenth International*

Conference on Knowledge Discovery and Data Mining, 374-383, Seattle, USA, August 2004.

[Zhong et al., 2003] Zhong, N., Yao, Y.Y., Ohshima, M., Peculiarity oriented multidatabase mining, *IEEE Transactions on Knowledge and Data Engineering*, 15(4), 952-960, 2003.