DATA MINING IN A LARGE DATABASE ENVIRONMENT

S. Y. Sung, K. Wang and B.L.Chua

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 109260

E-mail: ssung@iscs.nus.sg

ABSTRACT

Data mining, the process of discovering hidden and potentially useful information from very large databases, has been recognized as one of the most promising research topics in the 1990s. The essential problem faced in the mining of ***association rules*** is the generation of ***large items***, which are items that are present in at least s% (***minimal support***) of the total database tuples. As the large items and their counts information usually require much storage space, the ***minimal cover*** concept is introduced to achieve reductions in the storage size. ***Percentage contour***, an extension of minimal cover, is further introduced to aid in the handling of large databases.

1 INTRODUCTION

Data mining can be defined as the process of discovering hidden and potentially useful information from very large databases. It has been recognized as one of the most promising research topics in the 1990s by both database and machine learning researchers [2-10]. The purpose of database mining is to transform information, expressed in terms of stored data, into knowledge, expressed in terms of generalized statements, or rules about some characteristics, or relationships occurring in data.

Many important types of rules can be derived from databases. They include rules like classification, sequential patterns and similar sequences [5], etc. In [8], Agrawal et. al introduced a new class of rules called ***association rules***, and further provide the algorithm for finding such rules from a database. Given a database of sales transactions, association rules can be described as follows :

Discover all ***association rules*** such that in at least c% of transactions, the presence of an item (or a set of attributes) implies the occurrence of another item. The value c% is the ***confidence factor*** or ***confidence threshold*** of the rule. One such example is "If a customer buys bread and butter, then the customer also buys milk in at least 90% of the cases". The ***antecedent*** of this rule consists of bread and butter, while the ***consequent*** consists of milk alone.

Consider a supermarket with a large collection of items. Common business decisions that the management of the supermarket has to make include what items to put on sales, how to arrange the items on the shelves in order to maximize the profit, etc. [8]. Thus, association rules are considered important rules in the real world and have been widely applied to various databases. Applications on databases include customer behaviour analysis on sales data obtained from a large retailing company [8], a course enrollment database in a university [4] and a telephone company fault management database [4].

The essential problem of mining association rules is to find all items, that are present in at least s% of the total database tuples. The value s% is known as the ***minimal support***. It is important not to confuse support with confidence. While confidence is a measure of a rule's strength, support corresponds to statistical significance [8]. We refer to all items that have fractional transaction support above the minimal support as ***large items*** and all others as ***small items*** or ***non-large items***.

0-7803-3280-6/96/$5.00

When association rules are frequently probed by users, it is expensive to search the database to discover the rules every time. Alternatively, we can choose to store all the association rules. However, the number of association rules is usually much larger than the number of large items (as shown in [4]). Thus, it is still preferable to store the large items and their counts, rather than the association rules.

If there are only a few large items, storage of the large items and their counts do not cause much problem. However, once the number of large items becomes huge, the storage is no longer feasible. To solve the above problem, we introduce the concept of ***minimal covers***, which are able to reduce the amount of information stored. We further extend the concept of minimal covers and present the concept of ***percentage contours***, which can aid in the handling of large databases. Applications of both the minimal covers and percentage contours will be presented as well.

The rest of this paper is organized as follows. In Section 2, we will be giving a formal description of the underlying model. In Section 3, we present to readers the definition of minimal cover and its storage structure. By extending the concept of minimal covers, we further present the concept of percentage contour. In Section 4, we conduct four operations based on minimal covers and percentage contours. Concluding remarks will be given in Section 5.

2 DESCRIPTION OF MODEL

Let T be a database of transactions or tuples where each tuple t is represented as a binary vector. We would then have Attr(T), which is the set of all attributes in T. An ***item*** $I = \{I_1, I_2, ..., I_m\}$, where each element I_i, for $1 \le i \le m$, is an ***attribute***, is a nonempty subset of Attr(T). A tuple t is known to satisfy an item I if for all attributes I_i in I, $t[I_i] = 1$.

For a given item I, ***count***(I) or the count of item I, is the total number of tuples in the database that satisfy I. If there exists two items, X and Y, such that $X \subset Y$, then X is known as the ***parent*** or ***ancestor*** of Y. On the other hand, Y becomes the ***child*** or ***descendant*** of X. Also, if the number of different attributes between X and Y is m, then Y is known as the ***m-extension*** of X. Since X, being the parent of Y, consists of less attributes than Y, therefore count(X) ≥ count(Y) will always hold.

By an association rule, we mean an implication of the form $X \Rightarrow Y$. X and Y are both items which can consist of either a single attribute or a set of attributes. One point to note is that Y does not contain any common attributes as X (that is : $X \cap Y = \varnothing$). The rule $X \Rightarrow Y$ is satisfied in the set of tuples T with the confidence threshold $0 \le c \le 1$ iff at least c% of tuples in T that satisfy X also satisfy Y. We use the notation $X \Rightarrow Y \mid c$ to specify that the rule $X \Rightarrow Y$ has a confidence threshold of c, or simply $X \Rightarrow Y$, if no confusion arises. The rule $X \Rightarrow Y \mid c$ is also equivalent to $\frac{\text{count}(XY)}{\text{count}(X)} \ge c$. The ***support*** for a rule $X \Rightarrow Y$ is defined to be the fraction of tuples in T that satisfy $t[I_i] = 1$ for all I_i in XY.

Support of an item I is defined to be the fraction of tuples in the database that satisfy I. If we are interested only in those items that have their support exceeding a certain threshold, this support becomes the ***minimal support***. Therefore I is considered to be large if its support is greater than or equal to the minimal support.

As shown in [8], the problem of mining association rules can be decomposed into two subproblems :

- Generate all large items X, such that count(X) ≥ s * Card(T); where s is the minimal support and Card(T) is the total number of tuples in the database T.

- For each large item $X = I_1I_2 \ldots I_k$, $k \ge 2$, generate all possible association rules $Y \Rightarrow (X - Y)$, such that $Y \subset X$ and $1 \le$ ***length***$(Y) \le (k-1)$. (Note : length(Y) refers to the length of Y).

The ***length*** of an item is equivalent to the number of attributes it consists of. In general, an n-length item, or an item in ***level***-n, will consist of n attributes. Once the large items have been determined, the solution to the second subproblem becomes quite straightforward. In this paper however, we shall not

have any discussions on the determination of large items since we are placing our efforts on the introduction of minimal covers and its applications. Before we embark on the discussion of minimal covers, we would just like to mention that the large items generated from the database are stored according to their lengths.

3 CONCEPT OF MINIMAL COVERS

In this section, we shall introduce to readers the concept of ***minimal covers***, which can be used to reduce the amount of information or data stored. Such a reduction is particularly important when the databases concerned are distributed. The reason is : For distributed databases, we cannot avoid the possibility of transmitting data from one site to another. Since transmission of huge data takes time and causes much traffic over the network, it will be beneficial if we can transmit minimum possible data and yet not miss out on any important information. As a result, the concept of minimal covers is proposed.

3.1 FORMAL MODEL OF MINIMAL COVERS

An ***itemset*** is a set consisting of items. Given an itemset S, we define the ***generating set*** of S, g(S), as a collection of items in S and all their ancestors. g(S) is also called the closure of S. In turn, S is called the ***generator*** of g(S).

Definition 3.1. An itemset X is a ***cover*** of itemset S iff X is a subset of S and g(X)=g(S).
Definition 3.2. For itemset X to be a ***minimal cover*** of itemset S, X must be a cover of S and $X \subseteq Y$ for every cover Y of S.

Example. Given S = { I_1, I_2, I_3, I_4, I_1I_2, I_1I_3, I_1I_4, I_2I_3, $I_1I_2I_3$ }.
Let X = { I_1I_2, I_1I_3, I_1I_4, I_2I_3 }, Y = { I_3, I_4, I_1I_4, $I_1I_2I_3$ } and Z = { I_1I_4, $I_1I_2I_3$ }.
Itemset X is not a cover since item $I_1I_2I_3$ cannot be generated from X. Both itemsets Y and Z are covers, since all items present in S can be generated from them. However $Z \subset Y$, therefore itemset Y is not a minimal cover of S.

The minimal cover of itemset S, denoted as ***MC***(S), is actually the set of all ***maximal items*** of S. We call items that are not included in any other items of S as ***maximal items***. In another word, maximal items can never be the parents of any items in S. We present the following theorem to illustrate the content of a minimal cover.

Theorem 3.1. The minimal cover of itemset S is exactly the set of all maximal items of S.
Proof. Let M = set of all maximal items of S. For any arbitrary item x in S, x is either in M or is an ancestor of some item y in S. In the latter case, y is either in M or y is also an ancestor of some item z in S. The list goes on and eventually, the expansion will end at an item in M. This shows that x is either in M or is the ancestor of some item in M. Thus M is indeed a cover and therefore contains the minimal cover.
On the other hand, a cover of S must always contain M; otherwise it cannot be a cover. ❑

For any given generating set g(S), its corresponding minimal cover contains sufficient and necessary information to allow reconstruction of g(S). Thus, minimal covers are considered as the most compact subsets with respect to generating sets. To store a minimal cover MC, it is divided into subsets according to the lengths of its items. **Figure 3-1** shows the storage structure.

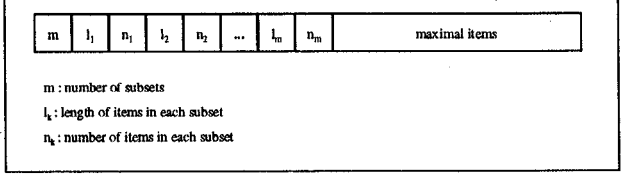

Figure 3-1 :
Storage Format of a Minimal Cover

3.2 PERCENTAGE CONTOUR

We shall now embark on the discussion of the extension of minimal covers, the ***percentage contour***. A ***contour*** is defined as a collection of itemsets $S_1, S_2, \ldots, S_n$, in which any two arbitrary itemsets S_i and S_j, $1 \le i \le n$, $1 \le j \le n$, is either $S_i \subseteq S_j$ or $S_j \subseteq S_i$.

Suppose we have a database T, with Card(T) referring to the number of tuples in T. For $0 \leq c \leq 1$, where c is any arbitrary support value, we define a ***percentage set*** as GE(c) = {all items in T which have counts $\geq c * Card(T)$}. Usually we have GE(0) = {all items in T} and GE(1) = $\emptyset$. When $c_1 > c_2$, $GE(c_2) - GE(c_1) = \{ X \mid X$ is an item in T and $c_1 * Card(T) > count(X) \geq c_2 * Card(T)\}$. For $1 > c_1 > c_2 > \ldots > c_n > 0$, the set PC = { $GE(c_i) \mid i = 1, 2, \ldots, n$ } forms the percentage contour of database T, where $GE(c_1) \subseteq GE(c_2) \ldots \subseteq GE(c_n)$.

Let a percentage contour PC = {$GE(c_1), \ldots, GE(c_n)$, where $c_1 > c_2 > \ldots > c_n$ }. To keep the storage space of PC small, we store only the minimal cover MC_i for each i, instead of the whole set of $GE(c_i)$. To further save on storage space, we store in PC the set $\Delta c_i = GE(c_i) - GE(c_{i-1})$, instead of $GE(c_i)$ itself. Now that we store only $\Delta c_i = GE(c_i) - GE(c_{i-1})$, we need to reconstruct the set $GE(c_i)$ using $GE(c_{i-1}) \cup \Delta c_i$.

The storage of the percentage contour is slightly modified from that of a minimal cover. The differences are : (i) there is an additional field p_i, indicating the lower percentage bound of a particular percentage set; (ii) the maximal items belonging to all the percentage sets are stored separately from the percentage set information. The modified storage format is shown in **Figure 3-2**.

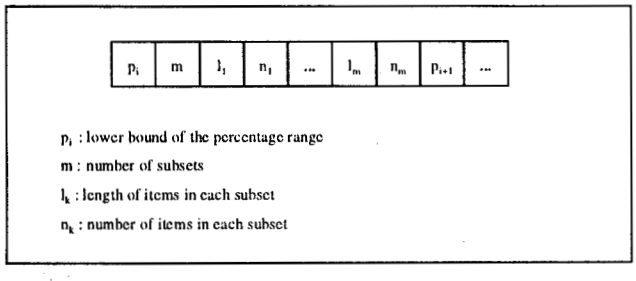

Figure 3-2 :
Storage Format of a Percentage Contour Information

4 OPERATIONS OF MINIMAL COVERS AND PERCENTAGE CONTOURS

Minimal cover, being the most compact way in representing a complete set, can be used to facilitate large distributed database mining in various ways. Four important and useful types of operations that can be applied to the minimal covers are discussed below. For discussion, assume that we have a database T that is distributed over n sites, with each site i having a fragment T_i. Let the nth site be the coordinator while the other sites are the participants.

4.1 UNION OPERATION

We are interested in finding for each site, all the large items that have a local support of at least c. After which we would like to compile the items from each site into a single result set.

First of all, all the participants have to transmit $S_i(c)$ = {all items in T_i that have count $\geq c * Card(T_i)$}. After receiving $S_i(c)$ from all participants, the coordinator will perform $S_U = \bigcup_{i=1}^{n} S_i(c)$, which includes its own set, $S_n(c)$. (Note : The transmission of all S_i sets, discussed in this section and later sections, are done using minimal covers, so that we can reduce the amount of data to be transmitted.)

The purpose of the union operation is not solely to combine all items collected from each site into a single set. It needs to identify items that may be redundant due to the presence of their descendants in the result set. Therefore, all items, except those that belong to the highest level, will be examined. Any items found to be covered by any higher level items will be excluded from the final result set. The end result is a union result set that is kept to the minimum possible size.

In [1], a new algorithm for determining the large items has been proposed. This algorithm brings into light another area whereby the union operation proves to be useful even in a centralized database.

Basically the algorithm executes in two phases. In the first phase, the database is horizontally divided into a number of non-overlapping partitions. The partitions are considered one at a time and all the large items for that partition are generated. At the

end of this phase, all the local large items for each partition are merged to generate a set of potential global large items. In the second phase, the actual support for these items are generated and the large items are identified.

After a brief look at the algorithm, we notice that the union operation is applicable at the end of the first phase, whereby the local large items from each partition are merged into a single set. This merging process is equivalent to the union operation, whereby all items from different sets are combined into a single union result set.

4.2 INTERSECTION OPERATION

Now we wish to find all large items that have global minimal support of at least c. Let us start off the discussion with n = 2. We have $S_1(c_1)$ = {all items in T_1 that have count $\geq c_1 * Card(T_1)$} and $S_2(c_2)$ = {all items in T_2 that have count $\geq c_2 * Card(T_2)$}. For any item X in $[S_1(c_1) \cap S_2(c_2)]$, it is true that $count(X) \geq [c_1 * Card(T_1) + c_2 * Card(T_2)]$. If $[c_1 * Card(T_1) + c_2 * Card(T_2)] \geq c * Card(T)$, then all items in $[S_1(c_1) \cap S_2(c_2)]$ are large items with respect to the support factor c.

In general, the coordinator will first collect $S_i(c_i)$ from participant i, $1 \leq i \leq n$, where $0 \leq c_i \leq 1$, and $\sum_{i=1}^{n} c_i * Card(T_i) = c * Card(T)$. (we select $c_i = c$ for all i = 1, 2, ..., n if we do not have any prior knowledge). The coordinator will generate two sets, $S_I = \bigcap_{i=1}^{n} S_i(c_i)$ and $S_C = \bigcap_{i=1}^{n} \overline{S_i(c_i)}$ and transmit them to the participants. (Note : The complement operation will be discussed in **Section 5.1.3**.) All items in S_I becomes part of the result set since they are already known to be large. However all items in S_C are ignored since they are already known to be small. Only the remaining items, that is those items that are neither in S_I nor in S_C, will be considered for further examination.

4.3 COMPLEMENT OPERATION

For this operation, given an itemset S_i, we are interested in finding all items that are not present in S_i. To simplify our discussion, assume we have a centralized database T with tuples that have four attributes, { I_1, I_2, I_3, I_4 }. With four attributes, we can form a total of $\binom{4}{1}+\binom{4}{2}+\binom{4}{3}+\binom{4}{4} = 4 + 6 + 4 + 1 = 15$ different combinations of attributes. For example, we have items I_1, I_2, I_3 and I_4 in level L_1 and a single item $I_1I_2I_3I_4$ in level L_4.

Suppose S_i now contains the following items : { I_1, I_2, I_3, I_1I_2, I_1I_3, I_2I_3 }. The complement set S_C is supposed to contain all possible combinations that are not present in S_i. In this case, S_C = { I_4, I_1I_4, I_2I_4, I_3I_4, $I_1I_2I_3$, $I_1I_2I_4$, $I_1I_3I_4$, $I_2I_3I_4$, $I_1I_2I_3I_4$ }.

To keep the complement result set small, we only need to store items that are not present in S_i and have length less than or equal to the maximum possible length in S_i. All other items with greater length are trivially known to be part of the complement set. Similar to the union operation, our complement result set is kept to the minimum possible size. Therefore, when the need arises for the above S_C to be transmitted to another site, only { I_4, I_1I_4, I_2I_4, I_3I_4 } need to be transmitted. All others { $I_1I_2I_3$, $I_1I_2I_4$, $I_1I_3I_4$, $I_2I_3I_4$, $I_1I_2I_3I_4$ } can be constructed by the receiver itself, provided it knows the maximum possible number of attributes (which is four in this case).

4.4 DIFFERENCE OPERATION

Given two sets S_1 and S_2, the difference result set $S_D = S_2 - S_1$. Basically, the difference operation is useful when given two itemsets, we are interested in collecting all items that are present in one set but not the other.

One application of the difference operation is : Refer to the example given in **Section 5.1.3**, we mentioned that the complement set S_C contains all items that are not present in a given set S_i. Assume we are to find the complement set for S_1, which contains { I_1, I_2, I_3, I_1I_2, I_1I_3, I_2I_3 }. Let S_2 contain all items with length less than or equal to maxLength, { I_1, ..., I_1I_2, ..., I_3I_4 }. By performing $S_D = S_2 - S_1$, we are able to obtain the complement set of S_1, that has been kept to the minimum possible size.

5 CONCLUSION

The main advantage of the minimal cover lies in the savings of storage space. In fact, savings on the storage space increases, as the minimal support value decreases. As we know that transmission of data from one site to another creates much traffic over the network, the decrease in data stored is especially beneficial when there is a need for transmission. We deduce that not only can minimal cover be applied to distributed database systems, which occur often in real-world applications, it is also attractive to a client/server environment.

By extending the concept of minimal covers, we introduce the ***percentage contour***, which is defined as a collection of percentage sets. Each set is just a collection of items that have counts greater than or equal to a certain percentage of the total database tuples. The percentage contour is economically stored as a set of minimal covers, in order to save storage space. The number of percentage sets in a percentage contour is only limited by the storage space available. As percentage contour can be used to facilitate large database mining, the more percentage sets it contains, the more accurate will be the results.

We further explore the various areas under which the minimal covers and the percentage contours proved to be useful. Four operations, namely union, intersection, complement and difference operations, which when applied on the percentage contours, helped us to achieve different purposes. For example, the intersection and complement operations together helped us to determine global large items for a distributed database. Another area of application is the compact storage of counts information, which facilitates the answering of the following 2 queries :

- Given an item X, estimates its count.
- Given a value $0 < c < 1$, search for all items X that have $count(X) \geq c * Card(T)$, where $Card(T)$ = total number of tuples in database T.

REFERENCES

[1] A. Savasere, E. Omiecinski and S. Navathe. "An Efficient Algorithm for Mining Association Rules in Large Databases". Proceedings of the 21st VLDB Conference, pages 432-444, Zurich, Switzerland, 1995.

[2] G. Piatetsky-Shapiro. Proceedings of AAAI-91 Workshop on Knowledge Discovery in Databases, July 1991, Anaheim, California.

[3] G. Piatetsky-Shapiro and W. Frawley. Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases, August 1989, Detroit, Michigan.

[4] H. Mannila, H. Toivonen and A. Inkeri Verkamo. "Efficient Algorithms for Discovering Association Rules". Proceedings of AAAI-94 Workshop on Knowledge Discovery in Databases, Eds. Usama M. Fayyad and Ramasamy Uthurusamy, pages 181-192, Seattle, Washington, July 1994.

[5] R. Agrawal. "Database Mining". Tutorial in DASFAA 95, April 1995, Singapore.

[6] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules". Proceedings of the 20th VLDB Conference, pages 487-499, Santiago, Chile, 1994.

[7] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer and A. Swami. "An Interval Classifier for Database Mining Applications". Proceedings of the 18th VLDB Conference, pages 560-573, Vancouver, British Columbia, Canada 1992.

[8] R. Agrawal, T. Imielinski and A. Swami. "Mining Association Rules Between Sets of Items in Large Databases". SIGMOD-93, pages 207-216, 1993, Washington, DC, USA.

[9] R. Agrawal, T. Imielinski and A. Swami. "Database Mining : A Performance Perspective". IEEE Transactions on Knowledge and Data Engineering, 5(6):914-925, December 1993.

[10] R. Krishnamurthy and T. Imielinski. "Practitioner Problems in Need of Database Research : Research Directions in Knowledge Discovery". SIGMOD Record, 20(3):76-78, September 1991.