Succinct Summarization of Transactional Databases: An Overlapped Hyperrectangle Scheme

Yang Xiang, Ruoming Jin, David Fuhry, Feodor F. Dragan
Department of Computer Science, Kent State University
Kent, OH, 44242, USA
{yxiang,jin,dfuhry,dragan}@cs.kent.edu

ABSTRACT

Transactional data are ubiquitous. Several methods, including frequent itemsets mining and co-clustering, have been proposed to analyze transactional databases. In this work, we propose a new research problem to succinctly summarize transactional databases. Solving this problem requires linking the high level structure of the database to a potentially huge number of frequent itemsets. We formulate this problem as a set covering problem using overlapped hyperrectangles; we then prove that this problem and its several variations are NP-hard. We develop an approximation algorithm $HYPER$ which can achieve a $ln(k) + 1$ approximation ratio in polynomial time. We propose a pruning strategy that can significantly speed up the processing of our algorithm. Additionally, we propose an efficient algorithm to further summarize the set of hyperrectangles by allowing false positive conditions. A detailed study using both real and synthetic datasets shows the effectiveness and efficiency of our approaches in summarizing transactional databases.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms, Theory

Keywords

hyperrectangle, set cover, summarization, transactional databases

1. INTRODUCTION

Transactional data are ubiquitous. In the business domain, from the world's largest retailers to the multitude of online stores, transactional databases carry the most fundamental business information: customer shopping transactions. In biomedical research, high-throughput experimental data, like microarray, can be recorded as transactional data, where each transaction records the conditions under which a gene or a protein is expressed [13] (or alternatively, repressed). In document indexing and search engine applications, a transactional model can be applied to represent the document-term relationship. Transactional data also appear in several different equivalent formats, such as binary matrix and bipartite graph, among others.

Driven by the real-world applications, ranging from business intelligence to bioinformatics, mining transactional data has been one of the major topics in data mining research. Several methods have been proposed to analyze transactional data. Among them, frequent itemset mining [2] is perhaps the most popular and well-known. It tries to discover sets of items which appear in at least a certain number of transactions. Recently, co-clustering [12], has gained much attention. It tries to simultaneously cluster transactions (rows) and items (columns) into different respective groups. Using binary matrix representation, co-clustering can be formulated as a matrix-factorization problem.

In general, we may classify transational data mining methods and their respective tools into two categories (borrowing terms from economics): *micro-pattern mining* and *macro-pattern mining*. The first type focuses on providing local knowledge of the transactional database, exemplified by frequent itemset mining. The second type works to offer a global view of the entire database; co-clustering is one such method. However, both types are facing some major challenges which significantly limit their applicability. On the micro-pattern mining side, the number of patterns being generated from the transaction data is generally very large, containing many patterns which differ only slightly from one another. Even though many methods have been proposed to tackle this issue, it remains a major open problem in the data mining research community. On the macro-pattern mining side, as argued by Faloutsos and Megalooikonomou [7], data mining is essentially the art of trying to develop concise descriptions of a complex dataset, and the conciseness of the description can be measured by Kolmogorov complexity. So far, limited efforts have been undertaken towards this goal of concise descriptions of transactional databases.

Above all, little work has been done to understand the relationship between the macro-patterns and micro-patterns. Can a small number of macro-pattern or high-level structures be used to infer or explain the large number of micro-patterns in a transactional database? How can the micro-patterns, like frequent itemsets, be augmented to form the macro-patterns? Even though this paper will not provide all the answers for all these questions, we believe the research problem formulated and addressed in this work takes a solid step in this direction, and particularly sheds light on a list of important issues related to mining transactional databases.

Specifically, we seek a succinct representation of a transactional database based on the *hyperrectangle* notation. A hyperrectangle is a Cartesian product of a set of transactions (rows) and a set of items (columns). A database is covered by a set of hyperrectangles if any element in the database, i.e., the transaction-item pair, is contained in at least one of the hyperrectangles in the set. Each hyperrectangle is associated with a representation cost, which is the sum of the representation costs (commonly the cardinality) of its set of transactions and set of items. The most succinct representation for a transactional database is the one which covers the entire database with the least total cost.

Here, the succinct representation can provide a high-level structure of the database and thus, mining succinct representation corresponds to a macro-pattern mining problem. In addition, the number of hyperrectangles in the set may serve as a measurement of the intrinsic complexity of the transactional database. In the meantime, as we will show later, the rows of the hyperrectangle generally correspond to the frequent itemsets, and the columns are those transactions in which they appear. Given this, the itemsets being used in the representation can be chosen as representative itemsets for the large collection of frequent itemsets, as they are more informative for revealing the underlying structures of the transactional database. Thus, the hyperrectangle notation and the succinct covering problem build a bridge between the macro-structures and the micro-structures of a transactional database.

1.1 Problem Formulation

Let the transaction database DB be represented as a binary matrix such that a cell (i,j) is 1 if a transaction i contains item j, otherwise 0. For convenience, we also denote the database DB as the set of all cells which are 1, i.e., $DB = \{(i,j) : DB[i,j] = 1\}$. Let the hyperrectangle H be the Cartesian product of a transaction set T and an item set I, i.e. $H = T \times I = \{(i,j) : i \in T$ and $j \in I\}$. Let $CDB = \{H_1, H_2, \cdots, H_p\}$ be a set of hyperrectangles, and let the set of cells being covered by CDB be denoted as $CDB^c = \bigcup_{i=1}^{p} H_i$.

If database DB is contained in CDB^c, $DB \subseteq CDB^c$, then, we refer to CDB as the *covering database* or the *summarization* of DB. If there is no false positive coverage in CDB, we have $DB = CDB^c$. If there is false positive coverage, we will have $|CDB^c \setminus DB| > 0$.

For a hyperrectangle $H = T \times I$, we define its cost to be the sum of the cardinalities of its transaction set and item set: $cost(H) = |T| + |I|$. Given this, the cost of CDB is

$$cost(CDB) = \sum_{i=1}^{p} cost(H_i) = \sum_{i=1}^{p} |T_i| + |I_i|$$

Typically, we store the transactional database in either horizontal or vertical representation. The horizontal representation can be represented as $CDB_H = \{\{t_i\} \times I_{t_i}\}$, where I_{t_i} is all the set of items transaction t_i contains. The vertical representation is as $CDB_V = \{T_j \times \{j\}\}$, where T_j is the transactions which contains item j. Let $\mathcal{T}$ be the set of all transactions in DB and $\mathcal{I}$ be the set of all items in DB. Then, the cost of these two representations are:

$$cost(CDB_H) = |\mathcal{T}| + \sum_{i=1}^{|\mathcal{T}|} |I_{t_i}| = |\mathcal{T}| + |DB|,$$

$$cost(CDB_V) = |\mathcal{I}| + \sum_{j=1}^{|\mathcal{I}|} |T_j| = |\mathcal{I}| + |DB|$$

In this work, we are interested in the following main problem. Given a transactional database DB and no false positive is allowed, how can we find the covering database CDB with minimal cost (or simply the *minimal covering database*) efficiently?

$$\min_{DB = CDB^c} cost(CDB)$$

In addition, we are also interested in how we can further reduce the cost of the covering database if false positive is allowed.

1.2 Our Contributions

Our contributions are as follows.

1. We propose a new research problem to succinctly summarize transactional databases, and formally formulate it as a variant of a weighted set covering problem based on a hyperrectangle notation.

2. We provide a detailed discussion on how this new problem is related to a list of important data mining problems (Section 2).

3. We study the complexity of this problem and prove this problem and its several variations are NP-hard (Section 3).

4. We develop an approximation algorithm $HYPER$ which can achieve a $ln(k) + 1$ approximation ratio in polynomial time. We also propose a pruning strategy that can significantly speed up the processing of our algorithm (Section 4).

5. We propose an efficient algorithm to further summarize the set of hyperrectangles by allowing false positive conditions. (Section 5).

6. We provide a detailed study using both real and synthetic datasets. Our research shows that our method can provide a succinct summarization of transactional data (Section 6).

2. RELATED RESEARCH PROBLEMS AND WORK

In this section, we discuss how the summarization problem studied in this work is related to a list of other important data mining problems, and how solving this problem can help to tackle those related problems.

Data Descriptive Mining and Rectangle Covering: This problem is generally in the line of descriptive data mining. More specifically, it is closely related to the efforts in applying rectangles to summarize underlying datasets. In [3], Agrawal et al. define and develop a heuristic algorithm to represent a dense cluster in grid data using a set of rectangles. Further, Lakshmanan et al. [11] consider the situation where a false positive is allowed. Recently, Gao et al. [8] extend descriptive data mining from a clustering description to a discriminative setting using a rectangle notation. Our problem is different from these problems from several perspectives. First, they focus on multi-dimensional spatial data where the rectangle area forms a continuous space. Clearly, the hyperrectangle is more difficult to handle because transactional data are discrete, so any combination of items or transactions can be selected to form a rectangle. Further, their cost functions are based on the minimal number of rectangles, whereas our cost is based on the cardinalities of sets of transactions and items. This is potentially much harder to handle.

Summarization for categorical databases: Data summarization has been studied by some researchers in recent years. Wang and Karypis proposed to summarize categorical databases by mining summary set [19]. Each summary set contains a set of summary itemsets. A summary itemset is the longest frequent itemsets supported by a transaction. This approach can be regarded as a special case of our summarization by fixing hyperrectangle width (i.e. the transaction dimension) to be one. Chandola and Kumar compress datasets of transactions with categorical attributes into informative representations by summarizing transactions [5]. They showed their methods are effective in summarizing network traffic. Their approach is similar to ours but different in the problem definition and research focus. Their goal is to effectively cover all transactions with more compaction gain and less information loss, while our goal is to effectively cover all cells (i.e. transaction item pair), which are finer granules of a database. In addition, our methods are shown to be effective not only by experimental results but also by the theoretical approximation bound.

Data Categorization and Comparison: Our work is closely related to the effort by Siebes *et al.* [15] [17] [18]. In [15] [17], they propose to recognize significant itemsets by their ability to compress a database based on the MDL principles. The compression strategy can be explained as covering the entire database using the non-overlapped hyperrectangles with no false positives allowed. The set of itemsets being used in the rectangles is referred to as the code table, and each transaction is rewritten using the itemsets in the code table. They try to optimize the description length of both the code table and the rewritten database. In addition, they propose to compare databases by the code length with regard to the same code table [18]. A major difference between our work and this work is that we apply *overlapped* hyperrectangles to cover the entire database. Furthermore, the optimization function is also different. Our cost is determined by the cardinalities of the sets forming the rectangles, and their cost is based on the MDL principle. In addition, we also study how the hyperrectangle can be further summarized by allowing false positive data. Thus, our methods can provide a much more succinct summarization of the transactional database. Finally, their approach is purely heuristic with no analytical results on the difficulty of their compression problem. As we will discuss in Section 3, we provide rigorous analysis and proof on the hardness of our summarization problem. We also develop an algorithm with proven approximation bound under certain constraints.

Co-clustering: As mentioned before, co-clustering attempts simultaneous clustering of both row and column sets in different groups in a binary matrix. This approach can be formulated as a matrix factorization problem [12]. The goal of co-clustering is to reveal the homogeneous block structures being dominated by either 1s or 0s in the matrix. From the summarization viewpoint, co-clustering essentially provides a so-called *checkerboard structure* summarization with false positive data allowed. Clearly, the problem addressed in this work is much more general in terms of the summarization structure and the false positive assumption (we consider both).

Approximate Frequent Itemset Mining: Mining *error-tolerant* frequent itemsets has attracted a lot of research attention over the last several years. We can look at error-tolerant frequent itemsets from two perspectives. On one side, it tries to recognize the frequent itemsets considering if some noise is added into the data. In other words, the frequent itemsets are disguised in the data. On another side, it provides a way to reduce the number of frequent itemsets since many of the frequent itemsets can be recognized as the variants of a true frequent itemset. This in general is referred to as pattern summarization [1][14]. Most of the efforts in error-tolerant frequent itemsets can be viewed as finding dense hyperrectangles with certain constraints. The support envelope notation proposed by Steinbach et al. [16] also fits into this framework. Generally speaking, our work does not directly address how to discover individual error-tolerant itemsets. Our goal is to derive a global summarization of the entire transactional database. However, we can utilize error-tolerant frequent itemsets to form a succinct summarization if false positives are allowed.

Data Compression: How to effectively compress large boolean matrices or transactional databases is becoming an increasingly important research topic as the size of databases is growing at a very fast pace. For instance, in [9], Johnson *et al.* tries to reorder the rows and columns so that the consecutive 1's and 0's can be compressed together. Our work differs because compression is concerned only with reducing data representation size; our goal is summarization, with aims to emphasize the important characteristics of the data.

3. HARDNESS RESULTS

In the following, we prove the complexity of the succinct summarization problem and several of its variants. We begin the problem with no false positives and extend it to false positive cases in corollary 1 and theorem 4. Even though these problems can quickly be identified as variants of the set-covering problem, proving them to be NP-hard is non-trivial as we need to show that at least one of the NP-hard problems can be reduced to these problems. Due to space limitations, we will only provide the major proof for the succinct summarization. The NP-hardness proof for its variants can be found in our technical report.

THEOREM 1. *Given DB, it is an NP-hard problem to construct a CDB of minimal cost which covers DB.*

Proof: To prove this theorem, we reduce the minimum set cover problem, which is NP-hard, to this problem.

The minimum set cover problem can be formulated as: Given a collection C of subsets of a finite set D, what is the minimum $|C'|$ such that $C' \subseteq C$ and every element in D belongs to at least one member of C'.

The reduction utilizes the database DB, whose entire set of items is D, i.e., each element in D corresponds to a unique item in DB. All items in a set $c \in C$ is recorded in $10|c|$ transactions in DB, denoted collectively as a set T_c. In addition, a special transaction w in DB contains all items in D. Clearly, this reduction takes polynomial time with respect to C and D. Note that we will assume that there is only one w in DB containing all items in D. If there were another one, it would mean there is a set c in C which covers the entire D; the covering problem could be trivially solved in that case. We will also assume each set c is unique in C.

Below we show that if we can construct a CDB with minimum cost, then we can find the optimal solution for the minimum set cover problem. This can be inferred by the following two key observations, which we state as lemmas 1 and 2.

LEMMA 1. *Let CDB be the minimal covering of DB. Then, all the T_c transactions in DB which record the same itemset $c \in C$ will be covered by a single hyperrectangle $T_i \times I_i \in CDB$, i.e. $T_c \subseteq T_i$ and $c = I_i$.*

LEMMA 2. *Let CDB be the minimal covering of DB. Let transaction w which contains all the items in D be covered by k hyperrectangles in CDB, $T_1 \times I_1, \cdots, T_k \times I_k$. Then each of the hyperrectangles is in the format of $T_c \cup \{w\} \times c$, $c \in C$. Further, the k itemsets in the hyperrectangles, $I_1, \cdots, I_k$, correspond to the minimum set cover of D.*

Putting these two lemmas together, we can immediately see that the minimal CDB problem can be used to solve the minimum set cover problem. Proofs of the two lemmas can be found in our technical report. □

Several variants of the above problem turn out to be NP-hard as well.

THEOREM 2. *Given DB, it is an NP-hard problem to construct a CDB with no more than k hyperrectangles that maximally covers DB.*

THEOREM 3. *Given DB and a budget δ, it is an NP-hard problem to construct a CDB that maximally covers DB with a cost no more than δ, i.e., $cost(CDB) \leq \delta$.*

COROLLARY 1. *When false positive coverage is allowed with $\frac{|CDB^c \setminus DB|}{|DB|} \leq \beta$, where β is a user-defined threshold, the above problems in theorems 1, 2, and 3 are still NP-complete.*

Assuming a set of hyperrectangles is given, i.e., the rectangles used in the covering database must be chosen from a predefined set, we can prove all the above problems are NP-hard as well.

THEOREM 4. *Given DB and a set S of candidate hyperrectangles such that $CDB \subseteq S$, it is NP-hard to 1) construct a CDB with minimal cost that covers DB; 2) construct a CDB to maximally cover DB with $|CDB| \leq k$; 3) construct a CDB to maximally cover DB with minimal cost where $cost(CDB) \leq \delta$, (δ is a user-defined budget). The same results hold for the false positive case: $\frac{|CDB^c \setminus DB|}{|DB|} \leq \beta$, where β is a user-defined threshold.*

4. ALGORITHMS FOR SUMMARIZATION WITHOUT FALSE POSITIVE

In this section, we develop algorithms to find minimal covering database CDB for a given transactional database with no false positives. As we mentioned before, this problem is closely related to the traditional weighted set covering problem. Let C be a candidate set of all possible hyperrectangles, which cover a part of the DB without false positive, i.e., $C = \{T_i \times I_i : T_i \times I_i \subseteq DB\}$. Then, we may apply the classical greedy algorithm to find the minimal set cover, which essentially correspond to the minimal covering database, as follows.

Let R be the covered DB (initially, $R = \emptyset$). For each possible hyperrectangle $T_i \times I_i \in C$, we define the price of H as:

$$\gamma(H) = \frac{|T_i| + |I_i|}{|T_i \times I_i \setminus R|}.$$

At each iteration, the greedy algorithm picks up the hyperrectangle H with the minimum $\gamma(H)$ (the cheapest price) and put it CDB. Then, the algorithm will update R accordingly, $R = R \cup T_i \times I_i$. The process continues until CDB completely covers DB ($R = DB$). It has been proved that the approximation ratio of this algorithm is $ln(k) + 1$, where k is the number of hyperrectangles in CDB [6].

Clearly, this algorithm is not a feasible solution for the minimal database covering problem due to the exponential number of candidate hyperrectangles in C, which in the worst case is in the order of $2^{|\mathcal{T}|+|\mathcal{I}|}$, where $\mathcal{T}$ and $\mathcal{I}$ are the sets of transactions and items in DB, respectively. To tackle this issue, we propose to work on a smaller candidate set, denoted as

$$C_\alpha = \{T_i \times I_i | I_i \in F_\alpha \cup I_s\},$$

where F_α is the set of all frequent itemsets with minimal support level α, and I_s is the set of all singleton sets (sets with only one item). We assume F_α is generated by Apriori algorithm. Essentially, we put constraint on the columns for the hyperrectangles. As we will show in the experimental evaluation, the cost of the minimal covering database tends to converge as we reduce the support level α. Note that this reduced candidate set is still very large and contains an exponential number of hyperrectangles. Let $T(I_i)$ be the transaction set where I_i appears. $|T(I_i)|$ is basically the support of itemset I_i. Then, the total number of hyperrectangles in C_α is

$$|C_\alpha| = \sum_{I_i \in F_\alpha \cup I_s} 2^{|T(I_i)|}.$$

Thus, even running the aforementioned greedy algorithm on this reduced set C_α is too expensive.

In the following, we describe how to generate hyperrectangles by an approximate algorithm which achieves the same approximation ratio with respect to the candidate set $|C_\alpha|$, while running in polynomial time in terms of $|F_\alpha \cup I_s|$ and $\mathcal{T}$.

4.1 The $HYPER$ Algorithm

As we mentioned before, the candidate set C_α is still exponential in size. If we directly apply the aforementioned greedy algorithm, it will take an exponential time to find the hyperrectangle with the cheapest price. The major challenge is thus to derive a polynomial-time algorithm that finds such a hyperrectangle. Our basic idea is to handle all the hyperrectangles with the same itemsets together as a single group. A key result here is we develop a polynomial time greedy algorithm which is guaranteed to find the hyperrectangle with the cheapest price among all the rectangles with the same itemsets. Since we only have $|F_\alpha \cup I_s|$ such groups, we can then find the globally cheapest rectangle in C_α in polynomial time.

Specifically, let $\overline{C_\alpha} = \{T(I_i) \times I_i\}$, $I_i \in F_\alpha \cup I_s$, where $T(I_i)$ is the set of all supporting transactions of I_i. We can see that C_α can easily be generated from $\overline{C_\alpha}$, which has only polynomial size $O(|(F_\alpha \cup I_s)|)$.

The sketch of this algorithm is illustrated in 1. Taking $\overline{C_\alpha}$ as input, the $HYPER$ algorithm repeatedly adds sub-hyperrectangles to set R. In each iteration (Lines 4-7), it will find the lowest priced sub-hyperrectangle H' from each hyperrectangle $T(I_i) \times I_i \in \overline{C_\alpha}$ (Line 4), and then select the cheapest H' from the set of selected sub-hyperrectangles (Line 5). H' will then be added into CDB (Line 6). Set R records the covered database DB. The process continues until CDB covers DB ($R = DB$, line 3).

Algorithm 1 HYPER(DB, $\overline{C_\alpha}$)

```
1: $R := \emptyset$;
2: $CDB := \emptyset$;
3: while $R \neq DB$ do
4:     call OptimalSubHyperRectangle to find $H'$ with minimum $\gamma(H')$
       for each $H_i = T(I_i) \times I_i \in \overline{C_\alpha}$;
5:     choose the $H'$ with minimum $\gamma(H')$ among all the ones discovered
       by OptimalSubHyperRectangle;
6:     $CDB \leftarrow CDB \bigcup \{H'\}$;
7:     $R \leftarrow R \bigcup H'$;
8: end while
9: return $CDB$
```

The key procedure is *OptimalSubHyperRectangle*, which will find the sub-hyperrectangle with the cheapest price among all the sub-hyperrectangles of $T(I_i) \times I_i$. Algorithm 2 sketches the procedure. The basic idea here is that we will decompose the hyperrectangle $T(I_i) \times I_i$ into *single-transaction* hyperrectangles $H_s = \{t_j\} \times I_i$ where $t_j \in T(I_i)$. Then, we will order those rectangles by the number of their uncovered cells (Lines $1-4$). We will perform an iterative procedure to construct the sub-hyperrectangle with cheapest price (Lines 6-13). At each iteration, we will simply choose the single-transaction hyperrectangle with maximal number of uncovered cells and try to add it into H'. If its addition can decrease $\gamma(H')$, we will add it to H'. By adding $H_s = \{t_j\} \times I_i$ into $H' = T_i \times I_i$, H' will be updated as $H' = (T_i \cup \{t_j\}) \times I_i$. We will stop when H_s begins to increase H'.

Algorithm 2 A Greedy Procedure to Find the Sub Hyper Rectangle with Cheapest Price

```
Procedure OptimalSubHyperRectangle($H$)
    {Input: $H = T(I_i) \times I_i$}
    {Output: $H' = T_i \times I_i, T_i \subseteq T(I_i)$}
 1: for all $H_s = \{t_j\} \times I_i \subseteq H$ do
 2:     calculate the number of uncovered cells in $H_s$, $|H_s \backslash R|$
 3: end for
 4: sort $H_s$ according to $|H_s \backslash R|$ and put it in $U$;
 5: $H' \leftarrow$ first hyperrectangle $H_s$ popped from $U$
 6: while $U \neq \emptyset$ do
 7:     pop a single-transaction hyperrectangle $H_s$ from $U$;
 8:     if adding $H_s$ into $H'$ increases $\gamma(H')$ then
 9:         break;
10:     else
11:         add $H_s$ into $H'$;
12:     end if
13: end while
14: return $H'$;
```

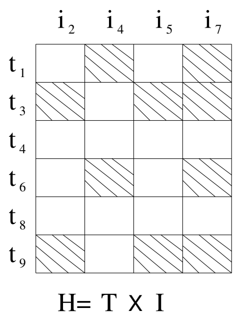


Figure 1: A hyperrectangle $H \in \overline{C_\alpha}$. Shaded cells are covered by hyperrectangles currently available in CDB.

Here is an example. Given hyperrectangle $H \in \overline{C_\alpha}$, consisting of $H = T(I) \times I = \{t_1, t_3, t_4, t_6, t_8, t_9\} \times \{i_2, i_4, i_5, i_7\}$, we construct H' with minimum $\gamma(H')$ in the following steps. First, we order all the single-transaction hyperrectangles according to their uncovered cells as follows: $\{t_4\} \times I$, $\{t_8\} \times I$, $\{t_1\} \times I$, $\{t_6\} \times I$, $\{t_3\} \times I$, $\{t_9\} \times I$. Beginning with $H' = \{t_4\} \times I$, the price $\gamma(H')$ is $(4+1)/4 = 5/4 = 1.25$.
If we add $\{t_8\} \times I$, $\gamma(H')$ falls to $\frac{5+1+0}{4+4} = \frac{6}{8} = 0.75$.
If we add $\{t_1\} \times I$, $\gamma(H')$ decreases to $\frac{6+1+0}{8+2} = \frac{7}{10} = 0.70$.
If we add $\{t_6\} \times I$, $\gamma(H')$ decreases to $\frac{7+1+0}{10+2} = \frac{8}{12} = 0.67$.
However, if we then add $\{t_3\} \times I$, $\gamma(H')$ would increase to $\frac{8+1+0}{12+1} = \frac{9}{13} = 0.69$. Therefore we stop at the point where $H' = \{t_4, t_8, t_1, t_6\} \times I$ and $\gamma(H') = 0.67$.

Properties of HYPER: We discuss several properties of HYPER, which will prove its approximation ratio.

LEMMA 3. *The* OptimalSubHyperRectangle *procedure finds the minimum $\gamma(H')$ for any input hyperrectangle $T(I_i) \times I_i \in \overline{C_\alpha}$.*

Proof: Let $H' = T_i \times I_i$ be the sub-hyperrectangle of $T(I_i) \times I_i$ with the least $\gamma(H')$. Then, we first prove that if a single-transaction $H_j = \{t_j\} \times I_i \subseteq H'$, then for any other single-transaction $H_k = \{t_k\} \times I_i, t_k \in T(I_i)$, if

$$|H_j \setminus R| \leq |H_k \setminus R|,$$

then H_k will be part of H'. By way of contradiction, without loss of generality, let us assume H_k is not in H'. Then, we have

$$\gamma(H') = \frac{|T_i| + |I_i|}{|H' \setminus R|} = \frac{|T_i \setminus \{t_j\}| + |I_i| + 1}{|(T_i \setminus \{t_j\}) \times I_i \setminus R| + |H_j \setminus R|}$$
$$= \frac{x+1}{y + |H_j \setminus R|} \geq \frac{x+1+1}{y + |H_j \setminus R| + |H_k \setminus R|}$$
$$(x = |T_i \setminus \{t_j\}| + |I_i|, y = |(T_i \setminus \{t_j\}) \times I_i \setminus R|),$$
$$\frac{x}{y} \geq \frac{x+1}{y + |H_j \setminus R|} \Rightarrow y \leq x|H_j \setminus R| \Rightarrow$$
$$(x+1)|H_k \setminus R| \geq y + |H_j \setminus R| \Rightarrow$$
$$(x+1)(y + |H_k \setminus R| + |H_j \setminus R|) \geq (x+2)(y + |H_j \setminus R|)$$

This shows that we can add H_k into H' to reduce the price ($\gamma(H')$). This contradicts the assumption that H' is the sub-hyperrectangle of $T(I_i) \times I_i$ with minimal cost. This suggests that we can find the lowest cost H' by considering the addition of single-transaction hyperrectangles in $T(I_i) \times I_i$, ordered by their number of uncovered cells. □

COROLLARY 2. *In OptimalSubHyperRectangle, if two single-transaction hyperrectangles with the same number of uncovered cell $|\{t_j\} \times I_i \backslash R| = |\{t_k\} \times I_k \backslash R|$, then either both of them can be added into H' or none of them.*

COROLLARY 3. *Assume that in iteration j of the* while loop *in the OptimalSubHyperRectangle procedure, we choose $H_j = \{t_j\} \times I_i$. We denote $a_j = |\{t_j\} \times I_i| \backslash R|$, and let $H' = T_i \times I_i$ with minimum $\gamma(H)$ contain the single-transaction hyperrectangles $H_1, H_2, \cdots, H_q$. Then we have $a_{q+1} < \frac{\sum_{i=1}^{q} a_i}{q + |I_i|}$.*

Proof: We know adding H_{q+1} to H' will increase $\gamma(H')$. Let $\gamma(H') = \frac{x}{y}$ before adding H_{q+1} into H'. According to the algorithm we have $\frac{x}{y} < \frac{x+1}{y + a_{q+1}}$, which means $a_{q+1} x < y$. We also know that $x = q + |I_i|$ and $y = \sum_{i=1}^{q} a_i$. Therefore $a_{q+1} < \frac{\sum_{i=1}^{q} a_i}{q + |I_i|}$. □

The above two corollaries can be used to speed up the OptimalSubHyperRectangle procedure. Corollary 2 suggests that we can process all the single-transaction hyperrectangles with the same number of uncovered cells as a single group. Corollary 3 can be used to quickly identify the cutting point for constructing H'.

Lemma 3 leads to the major property of the HYPER algorithm, stated as Theorem 5.

THEOREM 5. *The $HYPER$ algorithm has the exact same solution as the greedy approach for the weighted set covering problem and has $ln(k) + 1$ approximation ratio with respect to the candidate set C_α.*

Time Complexity of HYPER: Here we do not take it into account the time to generate F_α, which can be done through the classic Apriori algorithm. Assuming F_α is available, the $HYPER$ algorithm runs in $O(|\mathcal{T}|(|\mathcal{I}| + log|\mathcal{T}|)(|F_\alpha| + |\mathcal{I}|)k)$, where k is the number of hyperrectangles in CDB. The analysis is as follows. Assume the while loop in Algorithm 1 runs k times. Each time it chooses a H' with minimum $\gamma(H')$ from $\overline{C_\alpha}$, which contains no more than $|F_\alpha| + |\mathcal{I}|$ candidates. To construct H' with minimum $\gamma(H')$ for H, we need to update every single-transaction hyperrectangle in H, sort them and add them one by one, which takes $O(|\mathcal{T}||\mathcal{I}| + |\mathcal{T}|log|\mathcal{T}| + |\mathcal{T}|) = O(|\mathcal{T}|(|\mathcal{I}| + log|\mathcal{T}|))$ time. Since we need to do so for every hyperrectangle in $\overline{C_\alpha}$, it takes $O(|\mathcal{T}|(|\mathcal{I}| + log|\mathcal{T}|)(|F_\alpha| + |\mathcal{I}|))$. Therefore, the total time complexity is $O(|\mathcal{T}|(|\mathcal{I}| + log|\mathcal{T}|)(|F_\alpha| + |\mathcal{I}|)k)$. In addition, we note that k is bounded by $(|F_\alpha| + |\mathcal{I}|) \times |\mathcal{T}|$ since each hyperrectangle in $\overline{C_\alpha}$ can be visited at most $|\mathcal{T}|$ times. Thus, we conclude that our greedy algorithm performs in a polynomial time with respect to $|F_\alpha|$, $|\mathcal{I}|$ and $|\mathcal{T}|$.

4.2 Pruning Technique for HYPER

Although the time complexity of $HYPER$ is polynomial, it is still very expensive in practice since in each iteration, it needs to scan the entire $\overline{C_\alpha}$ to find the hyperrectangle with cheapest price. Theorem 6 reveals an interesting property of $HYPER$, which leads to an effective pruning technique for speeding up HYPER significantly (up to $|\overline{C_\alpha}| = |F_\alpha \cup \mathcal{I}|$ times faster!).

THEOREM 6. *For any $H \in \overline{C_\alpha}$, the minimum $\gamma(H')$ output by OptimalSubHyperRectangle will never decrease during the processing of the HYPER algorithm.*

Proof: This holds because the covered database R is monotonically increasing. Let R_i and R_j be the covered database at the i-th and j-th iterations in HYPER, respectively ($i < j$). Then, for any $H' = T_i \times I_i \subseteq T(I_i) \times I_i = H \in \overline{C_\alpha}$, we have

$$\gamma^i(H') = \frac{|T_i| + |I_i|}{T_i \times I_i \setminus R_i} \le \frac{|T_i| + |I_i|}{T_i \times I_i \setminus R_j} = \gamma^j(H'),$$

where $\gamma^i(H')$ and $\gamma^j(H')$ are the price for H' at iteration i and j, respectively. □

Algorithm 3 HYPER(DB,$\overline{C_\alpha}$)

```
1: $R \leftarrow \emptyset$;
2: $CDB \leftarrow \emptyset$;
3: call OptimalSubHyperRectangle to find $H'$ with minimum $\gamma(H')$ for
   each $T(I_i) \times I_i \in \overline{C_\alpha}$;
4: Sort all $T(I_i) \times I_i \in \overline{C_\alpha}$ into a queue $U$ according to their minimum
   $\gamma(H')$ from low to high and store $H'$ and its price (as the lower bound);
5: while $R \neq DB$ do
6:    Pop the first element $H_1$ with $H'_1$ from the queue $U$;
7:    call OptimalSubHyperRectangle to update $H'_1$ with minimum
      $\gamma(H'_1)$ for $H_1$;
8:    while $\gamma(H'_1) > \gamma(H'_2)$ do {$H_2$ is the next element in $U$ after popping the last hyper rectangle }
9:       insert $H_1$ with $H'_1$ back to $U$ in the sorting order;
10:      Pop the first element $H_1$ with $H'_1$ from the queue $U$;
11:      call OptimalSubHyperRectangle to update $H'_1$ with minimum
         $\gamma(H'_1)$ for $H_1$;
12:   end while
13:   $CDB \leftarrow CDB \cup \{H'_1\}$;
14:   $R \leftarrow R \bigcup H'_1$;
15:   call OptimalSubHyperRectangle to find the updated minimum
      $\gamma(H'_1)$ of $H_1$, and insert it back to the queue $U$ in the sorting order;
16: end while
17: return $CDB$;
```

Using Theorem 6, we can revise the $HYPER$ algorithm to prune the unnecessary visits of $H \in \overline{C_\alpha}$. *Simply speaking, we can use the minimum $\gamma(H')$ computed for H in the previous iteration as its lower bound for the current iteration since the minimum $\gamma(H')$ will be monotonically increasing over time.*

Our detailed procedure is as follows. Initially, we compute the minimum $\gamma(H')$ for each H in $\overline{C_\alpha}$. We then order all H into a queue U according to the computed minimum possible price ($\gamma(H')$) from the sub-hyperrectangle of H. To find the cheapest hyperrectangle, we visit H in the order of U. When we visit H, we call the *OptimalSubHyperRectangle* procedure to find the exact H' with the minimum price for H, and update its lower bound as $\gamma(H')$. We also maintain the current overall minimum price for the H visited so far. If at any point, the current minimum price is less than the lower bound of the next H in the queue, we will prune the rest of the hyperrectangles in the queue.

Algorithm 3 shows the complete HYPER algorithm which utilizes the pruning technique.

5. SUMMARIZATION OF THE COVERING DATABASE

In Section 4, we developed an efficient algorithm to find a set of hyperrectangles, CDB, to cover a transaction database. When false positive coverage is prohibited, the summarization is generally not succinct enough for the high-level structure of the transaction database to be revealed. In this section, we study how to provide more succinct summarization by allowing certain false positive coverage. Our strategy is to build a new set of hyperrectangles, referred to as the *succinct covering database* to cover the set of hyperrectangles found by HYPER. Let $SCDB$ be the set of hyperrectangles which covers CDB, i.e., for any hyperrectangle $H \in CDB$, there is a $H' \in SCDB$, such that $H \subseteq H'$. Let the false positive ratio of $SCDB$ be

$$\frac{|SCDB^c \backslash DB|}{|DB|},$$

where $SCDB^C$ is the set of all cells being covered by $SCDB$. Given this, we are interested the following two questions:

1. Given the false positive budget β, $\frac{|SCDB^c \backslash DB|}{|DB|} \le \beta$, how can we succinctly summarize CDB such that $cost(SCDB)$ is minimized?
2. Given $|SCDB| = k$, how can we minimize both the false positive ratio $\frac{|SCDB^c \backslash DB|}{|DB|}$ and the cost of $SCDB$?

We will focus on the first problem and we will show later that the same algorithm for the first problem can be employed for solving the second problem. Intuitively, we can lower the total cost by selectively merging two hyperrectangles in the covering set into one. We introduce the the merge operation ($\oplus$) for any two hyperrectangles, $H_1 = T_1 \times I_1$ and $H_2 = T_2 \times I_2$,

$$H_1 \oplus H_2 = (T_1 \cup T_2) \times (I_1 \cup I_2)$$

The net cost savings from merging H_1 and H_2 is

$$\gamma(H_1) + \gamma(H_2) - \gamma(H_1 \oplus H_2)$$

$$= |T_i| + |T_j| + |I_i| + |I_j| - |T_i \cup T_j| - |I_i \cup I_j|$$

To minimize $cost(CDB)$ with given false positive constraint $\frac{|CDB^c \backslash DB|}{|DB|} \le \beta$, we apply a greedy heuristic: *we will combine the hyperrectangles in CDB together so that the merge can yield the best savings with respect to the new false positive coverage, i.e., for any two hyperrectangles H_i and H_j,*

$$\arg\max_{H_i, H_j} \frac{|T_i| + |T_j| + |I_i| + |I_j| - |T_i \cup T_j| - |I_i \cup I_j|}{|(H_i \oplus H_j) \setminus SCDB^c|}.$$

Algorithm 4 sketches the procedure which utilizes the heuristics.

The second problem tries to group the hyperrectangles in CDB into k super- hyperrectangles. We can see the same heuristic can be employed to merge hyperrectangles. In essence, we can replace the *while* condition (Line 2) in Algorithm 4 with the condition that $SCDB$ has only k hyperrectangles. Finally, we note that the heuristic we employed here is similar to the greedy heuristic for the traditional *Knapsack problem* [10]. However, since we consider

Algorithm 4 HYPER+(DB, CDB, β)

```
1: $SCDB \leftarrow CDB$;
2: while $\frac{|SCDB^c \setminus DB|}{|DB|} \leq \beta$ do
3:     find the two hyper rectangles $H_i$ and $H_j$ in $SCDB$ whose merge is within the false positive budget:
           $$\frac{|(SCDB \setminus \{H_i, H_j\} \cup \{H_i \oplus H_j\})^c \setminus DB|}{|DB|} \leq \beta,$$
       and produces the maximum (or near maximum) saving-false positive ratio:
           $$\arg\max_{H_i, H_j} \frac{|T_i| + |T_j| + |I_i| + |I_j| - |T_i \cup T_j| - |I_i \cup I_j|}{|(H_i \oplus H_j) \setminus SCDB^c|}$$
4:     remove $H_i$ and $H_j$ from $SCDB$ and add $H_i \oplus H_j$: $SCDB \leftarrow SCDB \setminus \{H_i, H_j\} \cup \{H_i \oplus H_j\}$
5: end while
6: return $SCDB$;
```

only pair-wise merging, our algorithm does not have a guaranteed bound like the knapsack greedy algorithm. In addition, here we actually perform a random sampling merging in the experiments to speed up the algorithm. The full analysis will be available in our technical report. In Section 6, we show that our greedy algorithm works effectively for both real and synthetic transactional datasets.

6. EXPERIMENTAL RESULTS

In this section, we report our experimental evaluation on three real datasets and one synthetic dataset. All of them are publicly available from the FIMI repository [1]. The basic characteristics of the datasets are listed in Table 1. Borgelt's implementation of the well-known Apriori algorithm [4] was used to generate frequent itemsets. Our algorithms were implemented in C++ and run on Linux 2.6 on an AMD Opteron 2.2 GHz with 2GB of memory.

In our experimental evaluation, we will focus on answering the following questions.

1. How can HYPER (Algorithm 3) and HYPER+ (Algorithm 4) summarize a transactional dataset with respect to the summarization cost?
2. How can the false positive condition improve the summarization cost?
3. How does the set of frequent itemsets at different minimum support levels (α) affect the summarization results?
4. When users prefer a limited number of hyperrectangles, i.e. limited $|SCDB|$, how will the summarization cost and the false positive ratio $\frac{|SCDB^c \setminus DB|}{|DB|}$ look?
5. What is the running time of our algorithms?

To answer these questions, we performed a list of experiments, which we summarized as follows.

6.1 Summarization With Varying Support Levels

In this experiment, we study the summarization cost, the number of hyperrectangles, and the running time of HYPER and HYPER+ using the sets of frequent itemsets at different support levels.

In Figures 2(a) 2(b) 2(c), we show the summarization cost with respect to different support levels α on the chess, pumsb_star and T10I4D100K datasets. Each of these three figures has a total of six lines. Two of them are *reference lines*: the first reference line, named "DB", is the value of $|DB|$, i.e. the number of cells in DB. Recall that in the problem formulation, we denote $cost(CDB_H) = |\mathcal{T}| + |DB|$ and $cost(CDB_V) = |\mathcal{I}| + |DB|$. Thus, this reference line corresponds to the upper bound of any summarization cost. The other reference line, named "min_possible_cost", is the value of $|\mathcal{T}| + |\mathcal{I}|$. This corresponds to the lower bound any summarization can achieve, i.e. $SCDB$ contains only one hyperrectangle $\mathcal{T} \times \mathcal{I}$. The "CDB" line records the cost of CDB being produced by HYPER. The "SCDB_0.1", "SCDB_0.2", and "SCDB_0.4" lines record the cost of $SCDB$ being produced by HYPER+ with 10%, 20%, and 40% false positive budget.

Accordingly, in Figures 2(d) 2(e) 2(f), we show the number of hyperrectangles (i.e. k) in the covering database CDB or $SCDB$ at different support levels. The "CDB" line records $|CDB|$, and the "SCDB_0.1","SCDB_0.2", "SCDB_0.4" lines record $|SCDB|$ being generated by HYPER+ with 10%, 20%, and 40% false positive budget.

Figures 2(g) 2(h) 2(i) shows the running time. Here the line "CDB" records the running time of HYPER generating CDB from DB. The "SCDB-0.1", "SCDB-0.2", and "SCDB-0.4" lines record the running time of HYPER+ generating $SCDB$ under 10%, 20%, 40% false positive budget respectively. Here, we include both the time of generating CDB from DB (HYPER) and $SCDB$ from CDB (HYPER+). However, we do not count the running time of Apriori algorithm that is being used to generate frequent itemsets.

Here, we can make the following observations:

1. The summarization cost reduces as the support level α decreases; the number of hyperrectangles increases as the support level decreases; and the running time increases as the support level decreases. Those are understandable since the lower the support level is, the bigger the input ($\overline{C_\alpha}$) is for HYPER, and the larger the possibility for a more succinct covering database. However, this comes at the cost of a larger number of hyperrectangles.
2. The summarization cost and the number of hyperrectangles are dependent on the density of the database. HYPER and HYPER+ have a much smaller summarization cost with fewer hyperrectangles for the dense datasets, like chess, than for the sparse datasets, like pumsb_star. We believe this partly confirms our typical intuition that the high level structure of a dense transaction database can be relatively easy to describe. The frequent itemsets in the dense database can generally cover a much larger portion of the database, and thus, can serve as a good candidate to describe the high level structure of the database. However, the frequent itemsets in the sparse database will be more likely to span only a relatively small portion of the database. Thus, we will have to use a larger number of hyperrectangles to summarize the sparse database.
3. One of the most interesting observations is the "threshold behavior" and the "convergence behavior" across all the data, including the summarization cost, the number of hyperrectangles, and the running time on all these datasets. First, we observe the summarization cost tends to converge when α drops. Second, we can see that the number of hyperrectangles (k) increases rather sharply when α drops below some threshold, particularly for no false positive case (i.e. "CDB") and low false positive cases (i.e. "SCDB-0.1", "SCDB-0.2"), and the running time increases accordingly (sharing the same threshold). However, the convergence behavior tends to maintain the summarization cost at the same level or only decrease slightly. This we believe suggests that a lot of smaller hyperrectangles are chosen without reducing the cost significantly,

[1]http://fimi.cs.helsinki.fi/data/

Datasets	$\mathcal{I}$	$\mathcal{T}$	Avg. Len.	$\|DB\|$	density
chess	75	3,196	37	118,252	dense
pumsb_star	2,088	49,046	50.5	2,476,823	sparse
mushroom	119	8,124	23	186,852	dense
T10I4D100K	1,000	100,000	10	$\approx$ 1,000,000	sparse

Table 1: dataset characteristics

and that these small hyperrectangles are of little benefit to the data summarization. This phenomena suggests that a reasonably high α can produce a comparable summarization as a low α with much less computational cost, which would be especially important for summarizing very large datasets.

6.2 Summarization with Varying k

In this experiment, we will construct a succinct summarization with varying limited numbers of hyperrectangles (k). We perform the experiments on chess, mushroom and T10I4D100K datasets. We vary the number of k from around 100 to 10.

In Figures 2(m) 2(n) 2(o), each graph has two lines which correspond to two different minimum support levels α for generating $SCDB$. For instance, support_15 is the 15% minimal support for the HYPER+.

Here in Figures 2(j) 2(k) 2(l) , we observe that the summarization costs converge towards minimum possible cost when k decreases. This is understandable since the minimum possible cost is achieved when $k = 1$, i.e., there is only one hyperrectangle $\mathcal{T} \times \mathcal{I}$ in $SCDB$. In the meantime, we observe that the false positive ratio increases when k decreases. Especially, we observe a similar threshold behavior for the false positive ratio. This threshold again provides us a reasonable choice for the number of hyperrectangles to be used in summarizing the corresponding database.

We also observe that the sparse datasets, like T10I4D100K, tends to have a rather higher false positive ratio. However, if we compare with the worst case scenario, where only one hyperrectangle is used, the false positive ratio seems rather reasonable. For instance, the maximum false positive ratio is around 10000% for T10I4D100K, i.e., there is only around 1% ones in the binary matrix. Using the minimal support 0.5% and $k = 200$, our false positive ratio is less than 500%, which suggests that we use around 6% of the cells in the binary matrix to summarize T10I4D100K.

7. CONCLUSIONS

In this paper, we have introduced a new research problem to succinctly summarize transactional databases. We have formulated this problem as a set covering problem using overlapped hyperrectangles; we then proved that this problem and its several variations are NP-hard. We have developed two novel algorithms, $HYPER$ and $HYPER+$ to effectively summarize the transactional database. In the experimental evaluation, we have demonstrated the effectiveness and efficiency of our methods. In particular, we found interesting "threshold behavior" and "convergence behavior", which we believe can help us generate succinct summarizations in terms of the summarization cost, the number of hyperrectangles, and the computational cost. In the future, we plan to investigate those behaviors analytically and thus produce better summarizations. We also plan to apply this method on real world applications, such as microarray data in bioinformatics, for which we conjecture the hyperrectangles may correspond to certain biological process.

8. REFERENCES

[1] Foto N. Afrati, Aristides Gionis, and Heikki Mannila. Approximating a collection of frequent sets. In *KDD*, pages 12–19, 2004.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th VLDB Conf.*, September 1994.

[3] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.

[4] Christan Borgelt. Apriori implementation. http://fuzzy.cs.Uni-Magdeburg.de/ borgelt/Software. Version 4.08.

[5] Varun Chandola and Vipin Kumar. Summarization - compressing data into an informative representation. *Knowl. Inf. Syst.*, 12(3):355–378, 2007.

[6] V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res*, 4:233–235, 1979.

[7] Christos Faloutsos and Vasileios Megalooikonomou. On data mining, compression, and kolmogorov complexity. *Data Min. Knowl. Discov.*, 15(1):3–20, 2007.

[8] Byron J. Gao and Martin Ester. Turning clusters into patterns: Rectangle-based discriminative data description. In *ICDM*, pages 200–211, 2006.

[9] David Johnson, Shankar Krishnan, Jatin Chhugani, Subodh Kumar, and Suresh Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *VLDB'2004*, pages 13–23. VLDB Endowment, 2004.

[10] D. Pisinger Kellerer, Hans; U. Pferschy. *Knapsack Problems*. Springer Verlag, 2005.

[11] Laks V. S. Lakshmanan, Raymond T. Ng, Christine Xing Wang, Xiaodong Zhou, and Theodore J. Johnson. The generalized mdl approach for summarization. In *VLDB '02*, pages 766–777, 2002.

[12] Tao Li. A general model for clustering binary data. In *KDD*, pages 188–197, 2005.

[13] Sara C. Madeira and Arlindo L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 1(1):24–45, 2004.

[14] Jian Pei, Guozhu Dong, Wei Zou, and Jiawei Han. Mining condensed frequent-pattern bases. *Knowl. Inf. Syst.*, 6(5):570–594, 2004.

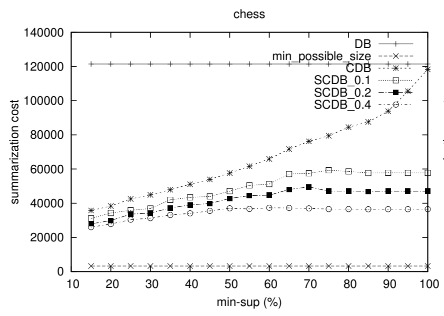
[15] Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. Item sets that compress. In *SDM*, 2006.

[16] Michael Steinbach, Pang-Ning Tan, and Vipin Kumar. Support envelopes: a technique for exploring the structure of association patterns. In *KDD '04*, pages 296–305, New York, NY, USA, 2004. ACM.

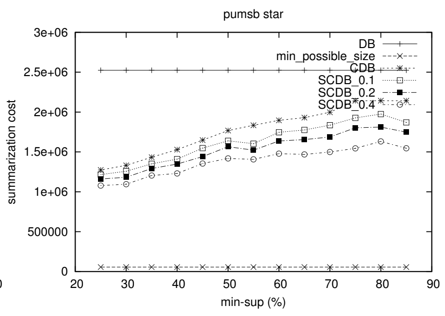
[17] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Compression picks item sets that matter. In *PKDD*, pages 585–592, 2006.

[18] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Characterising the difference. In *KDD '07*, pages 765–774, 2007.

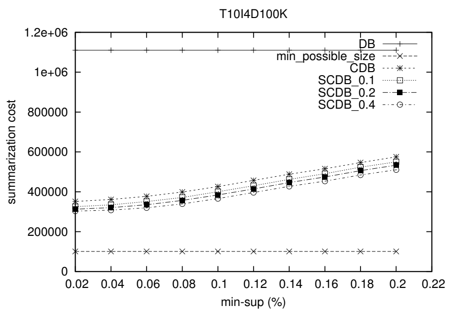
[19] Jianyong Wang and George Karypis. On efficiently summarizing categorical databases. *Knowl. Inf. Syst.*, 9(1):19–37, 2006.



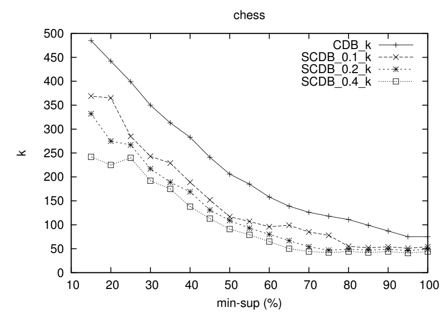
(a) chess cost



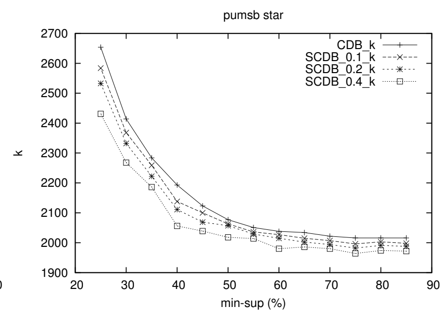
(b) pumsb_star cost



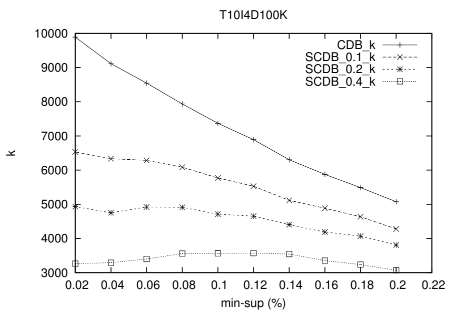
(c) T10I4D100K cost



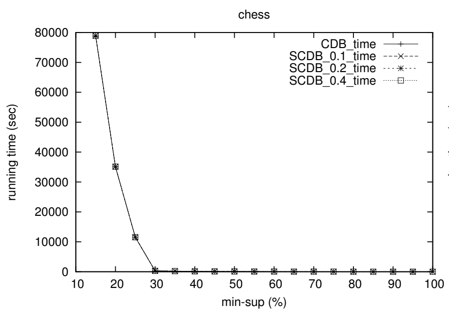
(d) chess k



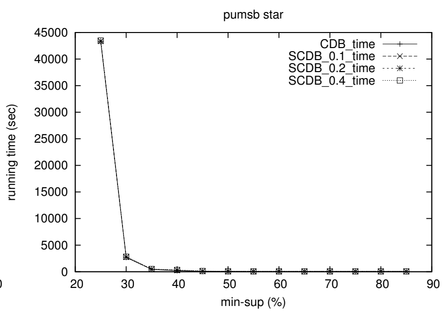
(e) pumsb_star k



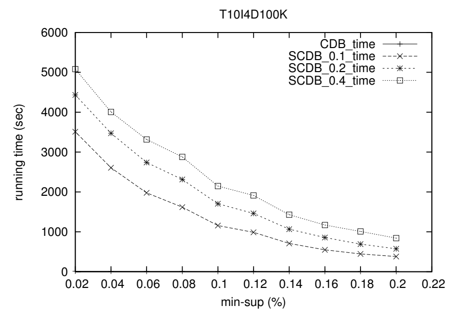
(f) T10I4D100K k



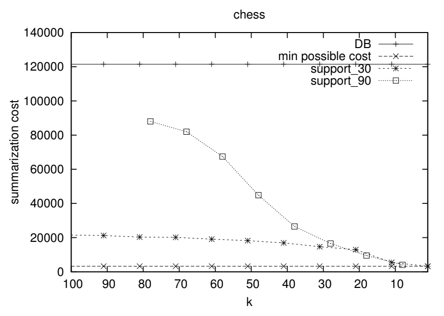
(g) chess running time



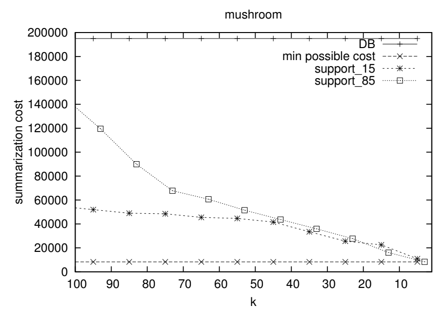
(h) pumsb_star running time



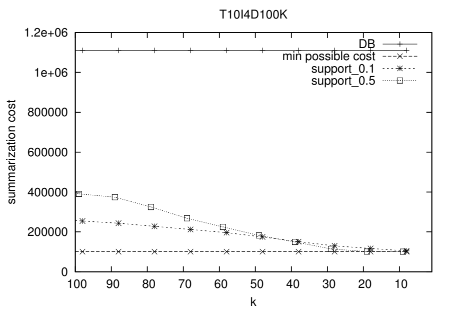
(i) T10I4D100K running time



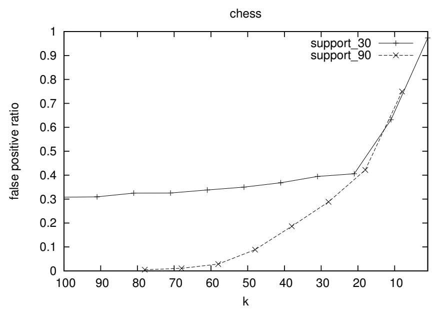
(j) chess cost



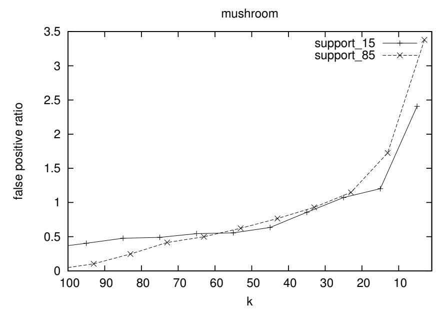
(k) mushroom cost



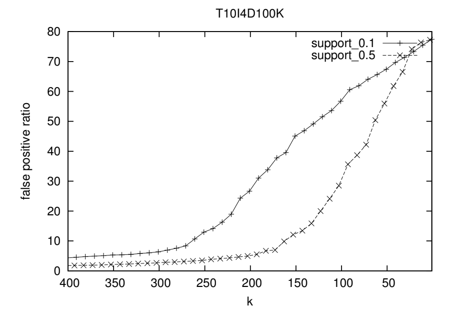
(l) T10I4D100K cost



(m) chess false positive ratio



(n) mushroom false positive ratio



(o) T10I4D100K false positive ratio

Figure 2: Experimental results