Data Min Knowl Disc (2011) 23:215–251
DOI 10.1007/s10618-010-0203-9

# Summarizing transactional databases with overlapped hyperrectangles

**Yang Xiang · Ruoming Jin · David Fuhry · Feodor F. Dragan**

Received: 13 January 2009 / Accepted: 20 September 2010 / Published online: 24 October 2010


**Abstract** Transactional data are ubiquitous. Several methods, including frequent itemset 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 (a concept generally regarded as tile according to some existing papers); we then prove that this problem and its several variations are NP-hard, and we further reveal its relationship with the

Responsible editor: Bart Goethals.

A preliminary version of this article appeared in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Xiang et al. 2008). This submission has substantially extended the previous paper and contains new and major-value added contribution in comparison with the conference publication.

Y. Xiang (✉)
Department of Biomedical Informatics, The Ohio State University, Columbus, OH 43210, USA
e-mail: yang.xiang@osumc.edu; yxiang@bmi.osu.edu

D. Fuhry
Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, USA
e-mail: fuhry@cse.ohio-state.edu

R. Jin · F. F. Dragan
Department of Computer Science, Kent State University, Kent, OH 44242, USA

R. Jin
e-mail: jin@cs.kent.edu

F. F. Dragan
e-mail: dragan@cs.kent.edu

compact representation of a directed bipartite graph. We develop an approximation algorithm HYPER which can achieve a logarithmic approximation ratio in polynomial time. We propose a pruning strategy that can significantly speed up the processing of our algorithm, and we also propose an efficient algorithm HYPER+ to further summarize the set of hyperrectangles by allowing false positive conditions. Additionally, we show that hyperrectangles generated by our algorithms can be properly visualized. A detailed study using both real and synthetic datasets shows the effectiveness and efficiency of our approaches in summarizing transactional databases.

**Keywords** Hyperrectangle · Tile · Set cover · Summarization · Transactional database · Frequent itemset mining

## 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 (Madeira and Oliveira 2004) (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 a binary matrix and a 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 (Agrawal and Srikant 1994) 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 (Hartigan 1972; Mirkin 1996; Li 2005), 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 transactional 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 (2007), 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-patterns 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* notion. 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.

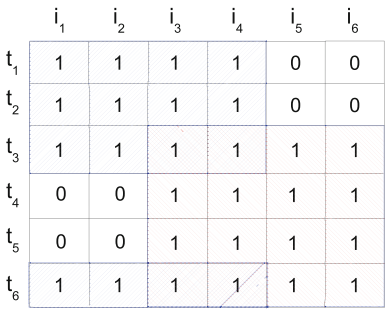
Here, the succinct representation can provide a high-level structure of the database and thus, mining a succinct representation corresponds to a macro-pattern mining problem. In addition, the number of hyperrectangles in the set may serve as a measurement of the intrinsic complexity of the transactional 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 notion 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 transactional 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. Throughout the paper, we use $\mathcal{T}$ to denote the set of all transactions of $DB$, and we use $\mathcal{I}$ to denote the set of all items of $DB$. For convenience, we also denote the database $DB$ as the set of all cells which are 1, i.e., $DB = \{(i, j) : DB[i, j] = 1\}$. To distinguish those zero cells in the binary matrix, a cell in $DB$ is also called as a *DB element*.

A *hyperrectangle* $H$ is a Cartesian product of a transaction set $T (T \subseteq \mathcal{T})$ and an item set $I (I \subseteq \mathcal{I})$, i.e. $H = T \times I = \{(i, j) : i \in T \text{ and } j \in I\}$. Let $CDB = \{H_1, H_2, \ldots, H_p\}$ 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 $DB$. Figure 1 gives a very simple example of transaction database $DB$ and its covering database $CDB$.



| | $i_1$ | $i_2$ | $i_3$ | $i_4$ | $i_5$ | $i_6$ |
|---|---|---|---|---|---|---|
| $t_1$ | 1 | 1 | 1 | 1 | 0 | 0 |
| $t_2$ | 1 | 1 | 1 | 1 | 0 | 0 |
| $t_3$ | 1 | 1 | 1 | 1 | 1 | 1 |
| $t_4$ | 0 | 0 | 1 | 1 | 1 | 1 |
| $t_5$ | 0 | 0 | 1 | 1 | 1 | 1 |
| $t_6$ | 1 | 1 | 1 | 1 | 1 | 1 |

**Fig. 1** A transaction database *DB*, with $\mathcal{T} = \{t_1, \ldots, t_6\}$ and $\mathcal{I} = \{i_1, \ldots, i_6\}$. $CDB = \{H_1, H_2\}$, where $H_1 = \{\{t_1, t_2, t_3, t_6\} \times \{i_1, i_2, i_3, i_4\}\}$ is shaded by *blue* and $H_2 = \{\{t_3, t_4, t_5, t_6\} \times \{i_3, i_4, i_5, i_6\}\}$ is shaded by *red*, is a covering database for *DB*. The cost of *CDB* is 16. (Color figure online)

If there is no false positive coverage in *CDB*, we have $CDB^c \subseteq DB$. If there is false positive coverage, we will have $|CDB^c \backslash DB| > 0$.

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: $cost(H) = |T| + |I|$. This definition is very intuitive as $|T| + |I|$ is the minimum number of integers (supposing each transaction or item has an integer ID) needed to present the Cartesian product $T \times I$. 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 contain 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:

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

and

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

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

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

Under certain constraint (e.g. $cost(CDB) \leq \delta$, or $|CDB| \leq k$), no $CDB$ may entirely cover $DB$. In these cases, we will be interested in finding a $CDB$ that can, under the given constraint, maximally cover $DB$, i.e., maximizing $|DB \cap CDB^c|$.

In addition, we are also interested in how we can further reduce the cost of covering the database if false positives are 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 notion.
2. We study the complexity of this problem and prove this problem and its several variations are NP-hard, and we show that our problem is closely related to another hard problem, the compact representation of a directed bipartite graph (Sect. 2).
3. We develop an approximation algorithm HYPER which can achieve a $ln(n) + 1$ approximation ratio in polynomial time. We also propose a pruning strategy that can significantly speed up the processing of our algorithm (Sect. 3).
4. We propose an efficient algorithm to further summarize the set of hyperrectangles by allowing false positive conditions (Sect. 4).
5. We show that hyperrectangles generated by our algorithms can be properly visualized (Sect. 5).
6. We provide a detailed discussion on how this new problem is related to a list of important data mining problems (Sect. 6).
7. We provide a detailed empirical study using both real and synthetic datasets. The empirical study shows that our method can not only succinctly summarize transactional data but also provide very useful information for people (Sect. 7).

## 2 The hardness of hyperrectangle summarization and its relationship with directed bipartite graph compression

In this section, we prove that hyperrectangle summarization problem and its several variations are NP-hard. In addition, we reveal the interesting link between hyperrectangle summarization and directed bipartite graph compression to further understand the nature of the hyperrectangle summarization problem.

### 2.1 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. We need to show that at least one of the NP-hard problems can be reduced to these problems.

**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 well-known as NP-hard, to this problem.

The minimum set cover problem can be formulated as follows: Given a collection $\mathcal{C}$ of subsets of a finite set $D$, find $\mathcal{C}' \subseteq \mathcal{C}$ with minimum $|\mathcal{C}'|$ such that every element in $D$ belongs to at least one member of $\mathcal{C}'$. We assume that there is no set $C \in \mathcal{C}$ that covers $D$ completely, otherwise the minimum set cover problem is too trivial to solve. We also assume each set $C$ is unique in $\mathcal{C}$.

To provide the reduction, we construct a transactional database *DB* from the set $D$ and $\mathcal{C}$. *DB* follows the transactional database definition in Sect. 1.1.

In our construction, each item in *DB* corresponds to an element in $D$. All items in a set $C \in \mathcal{C}$ are recorded in $2|C|$ (the reason to choose 2 will be clear in the following proof) transactions in *DB*, denoted collectively as a set $T_C (T_C \subseteq \mathcal{T})$ (for any two different items $C_1, C_2 \in \mathcal{C}$, $T_{C_1}$ and $T_{C_2}$ are disjoint). In addition, we create a special transaction $w$ in *DB* containing all items in $D$. Clearly, this reduction takes polynomial time with respect to $\mathcal{C}$ and $D$.

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 three key observations, which we state as Lemmas 1, 2, and 3.

**Lemma 1** *All transactions in $T_C$ will be covered by the same number of hyperrectangles in CDB.*

**Lemma 2** *Let CDB be the minimal covering of DB. Then, all the $T_C$ transactions in DB which record the same itemset $C \in \mathcal{C}$ 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 3** *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, $T_1 \times I_1, \ldots, T_k \times I_k$. Then each of the hyperrectangles is in the form of $T_C \cup \{w\} \times C$, $C \in \mathcal{C}$. Further, the $k$ itemsets in the hyperrectangles, $I_1, \ldots, I_k$, correspond to the minimum set cover of $D$.*

Putting these three lemmas together, we can see that the minimal *CDB* problem can be used to solve the minimum set cover problem. □

Proofs of the three lemmas are given below.

*Proof of Lemma 1:* Let $CDB(t_j), t_j \in T_C$ be the subset of *CDB*, which includes only the hyperrectangles covering transaction $t_j$, i.e., $CDB(t_j) = \{T_i \times I_i : t_j \in T_i\}$. Then, we conclude that for any two transactions $t_j$ and $t_l$ in $T_C$, $|CDB(t_j)| = |CDB(t_l)|$. This is true because if $|CDB(t_j)| > |CDB(t_l)|$, we can simply cover $t_j$ by $CDB(t_l)$ with less cost. This contradicts that *CDB* is the minimal covering database. □

*Proof of Lemma 2:* First, we prove that every transaction in $T_C$ is covered by one hyperrectangle, i.e. $|CDB(t_j)| = 1, t_j \in T_C$. According to Lemma 1, we can assume

every transaction in $T_C$ is covered by $k$ hyperrectangles in $CDB$. Assume $k > 1$. Let $CDB = \{T_1 \times I_1, \ldots, T_s \times I_s\}$ where $s \geq k$. Then, we can modify it as:

$$CDB' = \{(T_1 \setminus T_C) \times I_1, \ldots, (T_s \setminus T_C) \times I_s\} \cup \{T_C \times C\}.$$

Clearly, $CDB'$ covers $DB$, and the cost of $CDB'$ is smaller than the cost of $DB$:

$$\begin{aligned} cost(CDB') &= \sum_{i=1}^{s} (|T_i \setminus T_C| + |I_i|) + |T_C| + |C| \\ &= \sum_{i=1}^{s} (|T_i| + |I_i|) - k \times |T_C| + |T_C| + |C| \qquad |CDB(t_j)| = k, t_j \in T_C \\ &= \sum_{i=1}^{s} (|T_i| + |I_i|) + (3 - 2k) \times |C|) \qquad |T_C| = 2|C|, k > 1 \\ &< \sum_{i=1}^{s} (|T_i| + |I_i|) = cost(CDB). \end{aligned}$$

This is a contradiction. Thus, we can conclude that every transaction in $T_C$ can be covered by exactly one hyperrectangle.

We now show by contradiction that if more than one hyperrectangle is used to cover $T_C$, it cannot be minimal. Assume $T_C \times C$ is covered by $k$ hyperrectangles in $CDB$ where $k > 1$, expressed in the form of $T_C \times C \subseteq T_1 \times C \cup \cdots \cup T_k \times C$. We see that we can simply combine all $k$ of them into one: $T_1 \cup \cdots \cup T_k \times C$. The cost of that latter is less than the cost of the former: $|T_1 \cup \cdots \cup T_k| + |C| < \sum_{i=1}^{k} (|T_i| + |C|)$ This again contradicts the assumption that $CDB$ is the minimal database covering.

Put together, we can see that $T_C \times C$ is covered by only a single hyperrectangle. □

*Proof of Lemma 3:* Let $CDB(w) = \{T_1 \times I_1, \ldots, T_k \times I_k\}$. From Lemma 2, we can see all the transactions besides $w$ have been covered by a hyperrectangle $T_C \times C, C \in \mathcal{C}$. Thus, in $CDB(w)$, each hyperrectangle can either be in the form of $T_C \cup \{w\} \times C$, or $\{w\} \times I$, where $I \subseteq D$. We show that the latter case $\{w\} \times I$ where $I \subseteq D$ is not optimal. Assuming $I$ can be minimally covered by $z$ sets in $\mathcal{C}$, $I \subset C_1 \cup \cdots \cup C_z$, then $|I| \geq z$. Thus, we can cover $\{w\} \times I$ by $z$ hyperrectangles in the form of $(T_{C_1} \cup \{w\}) \times C_1, \ldots, (T_{C_z} \cup \{w\}) \times C_z$, with less cost. This contradicts that $CDB$ is minimal. Thus, we can see each hyperrectangle covering $w$ has the form $T_C \cup \{w\} \times C, C \in \mathcal{C}$.

Note that the cost of $CDB$ is

$$cost(CDB) = \sum_{C \in \mathcal{C}} (|T_C| + |C|) + s.$$

In the above equation, $s$ is minimized under the constraint that $I_1 \cup \cdots \cup I_s = D$. We conclude that $I_1, \ldots, I_s$ forms the minimum cover of $S$. □

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.*

*Proof* The result holds even for $k = 1$. To prove this theorem, we reduce the maximum edge biclique problem, which is NP-hard (Peeters 2003), to this problem with $k = 1$.

Maximum edge biclique problem can be formulated as: Given a bipartite graph $G = (V_1 \cup V_2, E)$, what is the biclique that has maximum edges?

The polynomial-time reduction is as follows: Create *DB* by letting $\mathcal{T} = V_1, \mathcal{I} = V_2$, and an element $(t, i)$ in *DB* ($t \in \mathcal{T}$ and $i \in \mathcal{I}$) if and only if $t$ and $i$ are the two end points of an edge in $E$. Also set $k = 1$.

Below we show that if we can construct a *CDB* with 1 Cartesian product that maximally covers *DB*, we find the maximum edge biclique in $G$.

Let the only Cartesian product in *CDB* be $T \times I$. If *CDB* maximally covers *DB*, then it is easy to see that $|T||I|$ is maximum. Because a transaction $t$ ($t \in T$) contains an item $i$ ($i \in I$) if and only if $t$ and $i$ are the two end points of an edge in $E$, we can conclude that we find the maximum edge biclique $T \cup I$ with $|T||I|$ edges. □

In view of Theorem 2 and by using the decision version of minimum set cover problem in the proof of Theorem 1, we can prove the following result.

**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*$(CDB) \leq \delta$.

If we reduce the above problems into false positive cases by letting $\beta = 0$, we can easily show the following corollary is correct.

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

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, it is NP-hard to*

(1) *construct a CDB* $\subseteq S$ *with minimal cost that covers DB;*
(2) *construct a CDB* $\subseteq S$ *to maximally cover DB with* $|CDB| \leq k$;
(3) *construct a CDB* $\subseteq S$ *to maximally cover DB with cost*$(CDB) \leq \delta$ ($\delta$ *is a user-defined budget).*

*The same results hold for the false positive case:* $\frac{|CDB^c \backslash DB|}{|DB|} \leq \beta$, *where β is a user-defined threshold.*

*Proof* Theorem 4 (1) can be deduced by letting the minimum set cover problem and *DB* be the same as in Theorem 1. For each $C \in \mathcal{C}$, we create two hyperrectangles in $S$, $\{(T_C \cup \{w\}) \times C\}$ and $\{T_C \times C\}$, where $T_C$ and $w$ are the same as in Theorem 1. It is easy to see that one of those hyperrectangles (but not both) will be selected for *CDB* according to this reduction. In view of the proof of Lemma 3, the solution will correspond to the set cover problem so that Theorem 4 (1) is correct.

In view of Theorem 2 and Theorem 3, it is easy to conclude Theorem 4 (2) and (3) are correct.

If we set $\beta = 0$, then the false positive cases reduce to the original statements. □

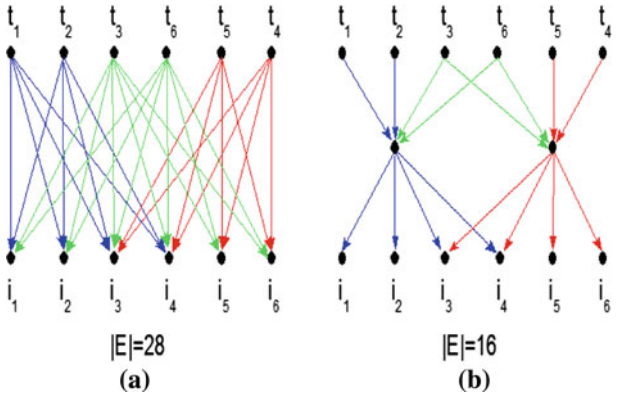



**Fig. 2** **a** A directed bipartite graph generated from the transaction database, **b** A directed graph compactly representing **a**

## 2.2 Relationship with compact representation of directed bipartite graph

Representing a directed bipartite graph by a compact directed graph is a fundamental problem related to important applications including the reachability query on DAG (Directed Acyclic Graph) (Agrawal et al. 1989) and modern coding theorem (Richardson and Urbanke 2008), and etc. Here, we reveal the close relationship between our summarization problem and compact representation of directed bipartite graph and show how the solutions for the former one can be applied to the latter one.

Consider a directed bipartite graph $G$ whose vertices can be divided into two sets $A$ and $B$. Any vertex in $A$ may point to any vertex in $B$.

Any graph reachability query scheme must be able to tell that $b$ ($b \in B$) can be reached from $a$ ($a \in A$) if there is an edge from $a$ to $b$ in the bipartite graph $G$. Further, we say graph $G'$ (not necessarily bipartite) is *reachability isomorphic* to $G$, if $A \subset V(G')$ and $B \subset V(G')$ such that if there is an edge from $a \in A$ to $b \in B$ in $G$ then there must be a directed path from $a \in A$ to $b \in B$ in $G'$, and vice versa.

The initial idea of compactly representing a directed bipartite graph by adding additional vertices was originally pointed out in Agrawal et al. (1989) as a simple heuristic for compressing transitive closure of a DAG. However, no detailed investigation was performed in Agrawal et al. (1989) and no further research is done on this topic to our best knowledge.

Considering a bipartite graph $G$ stored as a linked list, we may reduce the index size of reachability query on $G$ by finding a reachability isomorphic graph $G'$ of $G$ such that $|E(G')| < |E(G)|$. To get a $G'$, we can add some intermediate vertices, each of which points to a subset of $B$. Then vertices $A$ may point to some intermediate vertices so that $|E(G')|$ is less than $|E(G)|$. Figure 2 is an example: Graph (a) and graph (b) are reachability isomorphic but (b) has far fewer edges.

We can reduce the summarization problem of a transactional database to the problem of compactly representing a directed bipartite graph, and vice versa. Let each transaction of a database $DB$ be a vertex in the set $A$ of a directed bipartite graph $G$. Let each item in $\mathcal{I}$ of $DB$ be a vertex in the set $B$ of $G$. Then each hyperrectangle in the covering database $CDB$ is expressed as a new intermediate vertex in $G$.

For example, a transaction database in Fig. 1 has 28 elements if represented as $DB = \{(t_1, i_1), (t_1, i_2), \cdots\}$. But it only has 16 numbers if represented as $CDB = \{\{t_1, t_2, t_3, t_6\} \times \{i_1, i_2, i_3, i_4\}, \{t_3, t_4, t_5, t_6\} \times \{i_3, i_4, i_5, i_6\}\}$. Correspondingly, we can reduce the summarization scheme for the transaction database in Fig. 1 to the method of compactly representing a directed bipartite graph in Fig. 2.

## 3 Algorithms for summarization without false positives

In this section, we develop algorithms for summarizing transactional data by hyperrectangles without false positives. Starting with an intuitive greedy algorithm, we develop HYPER algorithm for summarizing transactional data by non-false-positive hyperrectangles in polynomial time with respect to $|\mathcal{I}|$, $|\mathcal{T}|$, and the size of frequent itemsets. Finally, we speedup HYPER algorithm by a pruning technique.

### 3.1 The intuitive greedy algorithm

In this section, we develop algorithms to find minimal cost 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 cover problem:

Given a collection $\mathcal{E}$ of subsets (each element $E \in \mathcal{E}$ has a cost $cost(E)$) of a finite set $D$, how can we cover $D$ with elements (i.e. subsets of $D$) in $\mathcal{E}$ such that the total cost is minimum?

As a Greedy algorithm for Weighted Set Cover problem (GWSC or GWSC algorithm in the following), at each step we choose a set $E \in \mathcal{E}$ which has the lowest price until $D$ is completely covered. The price of $E$ is defined as $cost(E)$ over the size of newly covered elements. Then the following theorem holds

**Theorem 5** **Chvátal (1979)** *The* GWSC *algorithm achieves an approximation ratio of* $\ln n + 1$ *over the optimal solution* (*i.e. minimum cost*) *where* $n$ *is the size of* $D$.

Let $\mathcal{C}$ be a candidate set of all possible hyperrectangles, which cover a part of the *DB* without false positives, i.e., $\mathcal{C} = \{T_i \times I_i : T_i \times I_i \subseteq DB\}$. Then, we may apply the GWSC algorithm to find the minimal set cover, which essentially corresponds to the minimal covering database, as follows.

Let $R$ be the covered *DB* (initially, $R = \emptyset$). For each possible hyperrectangle $H = T_i \times I_i \in \mathcal{C}$, we define its cost as $cost(H) = |T_i| + |I_i|$. Then the price of $H$ is

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

At each iteration, the greedy algorithm picks up the hyperrectangle $H$ with the minimum $\gamma(H)$ (the cheapest price) and inserts it into *CDB*. Then, the algorithm updates $R$ to be $R = R \cup T_i \times I_i$. The process continues until *CDB* completely covers *DB* ($R = DB$). According to Theorem 5, the approximation ratio of this algorithm is $\ln n + 1$, where $n = |DB|$. Recall that in Sect. 1.1 we define the transactional database

*DB* as the set of all cells which are 1, i.e., $DB = \{(i, j) : DB[i, j] = 1\}$. Hence, $|DB|$ is the total number of 1s in the binary matrix.

Clearly, this algorithm is not a feasible solution for the minimal database covering problem due to the exponential number of candidate hyperrectangles in $\mathcal{C}$, 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

$$\mathcal{C}_\alpha = \{T_i \times I_i : T_i \times I_i \subseteq DB, I_i \in \mathcal{F}_\alpha \cup \mathcal{I}\},$$

where $\mathcal{F}_\alpha$ is the set of all frequent itemsets with minimal support level $\alpha$, and $\mathcal{I}$ is the set of all singleton sets (sets with only one item). $\mathcal{F}_\alpha$ can be generated by Apriori algorithm (Agrawal et al. 1996). 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 $\alpha$. 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 $\mathcal{C}_\alpha$ is

$$|\mathcal{C}_\alpha| = \sum_{I_i \in \mathcal{F}_\alpha \cup \mathcal{I}} 2^{|T(I_i)|}.$$

Thus, even running the GWSC algorithm on this reduced set $\mathcal{C}_\alpha$ 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 $\mathcal{C}_\alpha$, while running in polynomial time in terms of $|\mathcal{F}_\alpha \cup \mathcal{I}|$ and $\mathcal{T}$.

### 3.2 The HYPER algorithm

As we mentioned before, the candidate set $\mathcal{C}_\alpha$ is still exponential in size. If we directly apply the GWSC 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 a novel greedy algorithm, which runs in polynomial time with respect to $|\mathcal{F}_\alpha|$, and is guaranteed to find the hyperrectangle with the cheapest price among all the hyperrectangles with the same itemsets. Since we only have $|\mathcal{F}_\alpha \cup \mathcal{I}|$ such groups, we can then find a globally cheapest hyperrectangle in $\mathcal{C}_\alpha$ in polynomial time.

Specifically, let $\mathcal{C}_\alpha^- = \{T(I_i) \times I_i : I_i \in \mathcal{F}_\alpha \cup \mathcal{I}\}$, where $T(I_i)$ is the set of all supporting transactions of $I_i$. We can see that $\mathcal{C}_\alpha$ can easily be generated from $\mathcal{C}_\alpha^-$, which has only polynomial size $O(|(\mathcal{F}_\alpha \cup \mathcal{I})|)$.

The sketch of this algorithm is illustrated in Algorithm 1. Taking $\mathcal{C}_\alpha^-$ as input, the HYPER algorithm repeatedly adds *sub-hyperrectangles* to set $R$. In this paper, we say a hyperrectangle $H_1 = \{T_1 \times I_1\}$ is a *sub-hyperrectangle* of $H_2 = \{T_2 \times I_2\}$, if and only if $T_1 \subseteq T_2$ and $I_1 = I_2$.

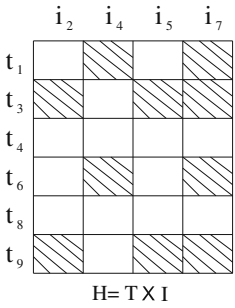



**Fig. 3** A hyperrectangle $H \in \mathcal{C}_\alpha^-$. Shaded cells are *DB* elements covered by hyperrectangles currently available in *CDB*

In each iteration (Lines 4–9), HYPER will find the lowest priced sub-hyperrectangle $H'$ from each hyperrectangle $T(I_i) \times I_i \in \mathcal{C}_\alpha^-$ (Line 4-6), and then select the cheapest $H'$ from the set of selected sub-hyperrectangles (Line 7). $H'$ will then be added into *CDB* (Line 8). Set $R$ records the covered database *DB*. The process continues until *CDB* covers *DB* ($R = DB$, line 3).

**Algorithm 1** HYPER($DB, \mathcal{C}_\alpha^-$)

1: $R \leftarrow \emptyset$;
2: $CDB \leftarrow \emptyset$;
3: **while** $R \neq DB$ **do**
4: **for all** $H_i \in \mathcal{C}_\alpha^-$ **do**
5: $X_i$ =FINDHYPER($H_i$,$R$);
6: **end for**
7: $H' = \arg\min_{X_i} \gamma(X_i)$;
8: $CDB \leftarrow CDB \cup \{H'\}$;
9: $R \leftarrow R \cup H'$;
10: **end while**
11: **return** *CDB*

The key procedure is FINDHYPER, 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 $\mathcal{S} = \{t_j\} \times I_i$, where $t_j \in T(I_i)$. Then, we will order those rectangles by the number of uncovered *DB* elements they contain (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 *DB* elements and try to add it into $H'$. If its addition can decrease $\gamma(H')$, we will add it to $H'$. By adding $\mathcal{S} = \{t_j\} \times I_i$ 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 $\mathcal{S}$ begins to increase $H'$.

*Example 1* (Fig. 3) Given hyperrectangle $H \in \mathcal{C}_\alpha^-$, 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 *DB* elements 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 $(4+1)/4 = 5/4 = 1.25$.

**Algorithm 2** A greedy procedure to find the sub-hyperrectangle with cheapest price

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

Adding $\{t_8\} \times I$ into $H'$, $\gamma(H')$ falls to $\frac{5+1+0}{4+4} = \frac{6}{8} = 0.75$.
Adding $\{t_1\} \times I$ into $H'$, $\gamma(H')$ decreases to $\frac{6+1+0}{8+2} = \frac{7}{10} = 0.70$.
Adding $\{t_6\} \times I$ into $H'$, $\gamma(H')$ decreases to $\frac{7+1+0}{10+2} = \frac{8}{12} = 0.67$.
However, if we then add $\{t_3\} \times I$ into $H'$, $\gamma(H')$ 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 4** *The* FINDHYPER *procedure finds a sub-hyperrectangle $H'$ with minimum $\gamma(H')$ for any input hyperrectangle $T(I_i) \times I_i \in \mathcal{C}_\alpha^-$.*

*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 claim that if there exists two single-transaction hyperrectangles $H_j$ and $H_l$ such that $H_j = \{t_j\} \times I_i \subseteq H'$, $H_l = \{t_l\} \times I_i$, $t_l \in T(I_i)$, and $|H_j \setminus R| \leq |H_l \setminus R|$ (i.e. $H_l$ covers the same or larger number of uncovered elements), then $H_l$ will be part of $H'$. We prove the above claim in the following:

Let

$$\gamma(H') = \frac{|T_i| + |I_i|}{|H' \backslash R|} = \frac{|T_i \backslash \{t_j\}| + |I_i| + 1}{|(T_i \backslash \{t_j\}) \times I_i \backslash R| + |H_j \backslash R|}$$

and

$$x = |T_i \backslash \{t_j\}| + |I_i|, \quad y = |(T_i \backslash \{t_j\}) \times I_i \backslash R|.$$

Then we conclude

$$\gamma(H') = \frac{x + 1}{y + |H_j \backslash R|}.$$

Because $\gamma(H')$ is minimum, we have

$$\frac{x}{y} \geq \frac{x+1}{y+|H_j \setminus R|},$$

which implies

$$y \leq x|H_j \setminus R|.$$

Because $|H_j \setminus R| \leq |H_l \setminus R|$, we have

$$(x+1)|H_l \setminus R| \geq y + |H_j \setminus R|,$$

which implies

$$(x+1)(y+|H_l \setminus R| + |H_j \setminus R|) \geq (x+2)(y+|H_j \setminus R|),$$

i.e.,

$$\frac{x+1}{y+|H_j \setminus R|} \geq \frac{x+1+1}{y+|H_j \setminus R|+|H_l \setminus R|}.$$

It means that adding $H_l$ into $H'$ can reduce the price $\gamma(H')$, a contradiction to the assumption that $H'$ is the sub-hyperrectangle of $T(I_i) \times I_i$ with minimal cost.

Since any two single-transaction hyperrectangles in $H'$ are disjoint, the above claim suggests that FINDHYPER procedure finds 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 *DB* elements. □

**Corollary 2** *In* FINDHYPER*, if two single-transaction hyperrectangles have the same number of uncovered DB elements, i.e.,* $|\{t_j\} \times I_i \setminus R| = |\{t_l\} \times I_i \setminus R|$*, then either both of them can be added into* $H'$ *or none of them.*

*Proof* Without loss of generality assume in the HYPER algorithm a single-transaction hyperrectangle $H_{s_{j_1}}$ is ranked before another single-transaction hyperrectangle $H_{s_{j_2}}$, and $|H_{s_{j_1}} \setminus R| = |H_{s_{j_2}} \setminus R| = a$. Before adding $H_{s_{j_1}}$ into $H'$ where $H' = T_i \times I_i$, $\gamma(H') = \frac{|T_i|+|I_i|}{|H' \setminus R|}$. Let $x = |T_i| + |I_i|$, $y = |H' \setminus R|$, then $\gamma(H') = \frac{x}{y}$. Since $\frac{x}{y} < \frac{x+1}{y+a}$ implies $\frac{x+1}{y+a} < \frac{x+2}{y+2a}$, and $\frac{x}{y} \geq \frac{x+1}{y+a}$ implies $\frac{x+1}{y+a} \geq \frac{x+2}{y+2a}$, we conclude that either both $H_{s_{j_1}}$ and $H_{s_{j_2}}$ are added into $H'$ or none of them. □

Corollary 2 suggests that we can process all the single-transaction hyperrectangles with the same number of uncovered elements as a single group so as to speed up the FINDHYPER procedure.

The following corollary (Corollary 3) provides a formula to quickly identify the cutting point for constructing $H'$. After calculating the coverage for each single-transaction hyperrectangle and ranking them according to their coverages in descending

order $(H_1, H_2, \ldots)$, we can quickly identify the number $q$ such that $H'$ is composed of $q$ single-transaction hyperrectangles $(H_1, H_2, \ldots, H_q)$.

**Corollary 3** *Assume that in iteration $j$ of the while loop in the* FINDHYPER *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 $a_{q+1} < \frac{\sum_{i=1}^{q} a_i}{q+|I_i|}$. □

Lemma 4 and the GWSC algorithm (Chvátal 1979) (with log approximation bound) lead to the major property of the HYPER algorithm, stated as Theorem 6.

**Theorem 6** *For our problem of finding the minimum cost CDB to cover DB, the solution of* HYPER *has exactly the same approximation ratio as the solution of* GWSC *given candidate set $\mathcal{C}_\alpha$. In other words,* HYPER *has $ln(n)+1 (n = |DB|)$ approximation ratio with respect to the optimal solution given candidate set $\mathcal{C}_\alpha$.*

*Proof* If the following claim is correct, by literally following the proof of Theorem 5, we can easily show Theorem 6 is correct.

Claim: Given *DB*, $R$ ($R \neq DB$), and $\mathcal{C}_\alpha$, letting $H_1 = \{T_1 \times I_1\}$ be the hyperrectangle discovered by the first selection of HYPER (i.e., step 4 to step 7 of the first while loop), and letting $H_2 = \{T_2 \times I_2\}$ be the hyperrectangle returned by the first selection of GWSC for weighted set cover, We claim that $\gamma(H_1) = \gamma(H_2)$.

To prove this claim, we first show that $\gamma(H_1) \geq \gamma(H_2)$. This is true because any output by FINDHYPER is a hyperrectangle in $\mathcal{C}_\alpha$, while GWSC will always select a minimum price hyperrectangle in $\mathcal{C}_\alpha$.

Second, we show that $\gamma(H_1) \leq \gamma(H_2)$. According to the definitions of $\mathcal{C}_\alpha$ and $\mathcal{C}_\alpha^-$, there must exist a hyperrectangle $H_3 = \{T_3 \times I_3\} \in \mathcal{C}_\alpha^-$ such that $I_2 = I_3$ and $T_2 \subseteq T_3$. If $\gamma(H_1) > \gamma(H_2)$, according to Lemma 4, first selection of HYPER should return a hyperrectangle with price no more than $\gamma(H_2)$, a contradiction. This completes the proof. □

**Time Complexity of HYPER:** Here we do not take into account the time to generate $\mathcal{F}_\alpha$, which can be done through the classic Apriori algorithm. Assuming $\mathcal{F}_\alpha$ is available, the HYPER algorithm runs in $O(|\mathcal{T}|(|\mathcal{I}|+\log|\mathcal{T}|)(|\mathcal{F}_\alpha|+|\mathcal{I}|)k)$, 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 $\mathcal{C}_\alpha^-$, which contains no more than $|\mathcal{F}_\alpha| + |\mathcal{I}|$ 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 $\mathcal{C}_\alpha^-$, it takes $O(|\mathcal{T}|(|\mathcal{I}| + \log|\mathcal{T}|)(|\mathcal{F}_\alpha| + |\mathcal{I}|))$. Therefore, the total time complexity is $O(|\mathcal{T}|(|\mathcal{I}| + \log|\mathcal{T}|)(|\mathcal{F}_\alpha| + |\mathcal{I}|)k)$. In addition, we note that $k$ is bounded by

$(|\mathcal{F}_\alpha| + |\mathcal{I}|) \times |\mathcal{T}|$ since each hyperrectangle in $\mathcal{C}_\alpha^-$ can be visited at most $|\mathcal{T}|$ times. Thus, we conclude that HYPER, our novel greedy algorithm, runs in polynomial time with respect to $|\mathcal{F}_\alpha|$, $|\mathcal{I}|$ and $|\mathcal{T}|$.

### 3.3 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 $\mathcal{C}_\alpha^-$ to find the hyperrectangle with cheapest price. Theorem 7 reveals an interesting property of HYPER, which leads to an effective pruning technique for speeding up HYPER significantly (up to $|\mathcal{C}_\alpha^-| = |\mathcal{F}_\alpha \cup \mathcal{I}|$ times faster).

**Theorem 7** *For any $H \in \mathcal{C}_\alpha^-$, the minimum $\gamma(H')$ output by* FINDHYPER *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 $H' = T_i \times I_i \subseteq T(I_i) \times I_i = H \in \mathcal{C}_\alpha^-$, we have

$$\gamma(H', R_i) = \frac{|T_i| + |I_i|}{T_i \times I_i \setminus R_i} \leq \frac{|T_i| + |I_i|}{T_i \times I_i \setminus R_j} = \gamma(H', R_j).$$

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

**Algorithm 3** HYPER($DB$,$\mathcal{C}_\alpha^-$)

```
1: R ← ∅;
2: CDB ← ∅;
3: call FINDHYPER to find H' with minimum γ(H') for each T(I_i) × I_i ∈ C_α^-;
4: Sort all T(I_i) × I_i ∈ C_α^- into a queue U according to their minimum γ(H') from low to high and store
   H' and its price (as the lower bound);
5: while R ≠ DB do
6:    Pop the first element H_1 with H_1' from the queue U;
7:    call FINDHYPER to update H_1' with minimum γ(H_1') for H_1;
8:    while γ(H_1') > γ(H_2') do {H_2 is the next element in U after popping the last hyperrectangle }
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 FINDHYPER to update H_1' with minimum γ(H_1') for H_1;
12:   end while
13:   CDB ← CDB ∪ {H_1'};
14:   R ← R ⋃ H_1';
15:   call FINDHYPER to find the updated minimum γ(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 7, we can revise the HYPER algorithm to prune the unnecessary visits of $H \in \mathcal{C}_\alpha^-$. 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. It is necessary to point out that similar pruning techniques have appeared in early literature (see Minoux (1977) for an example).

Our detailed procedure is as follows. Initially, we compute the minimum $\gamma(H')$ for each $H$ in $\mathcal{C}_\alpha^-$. 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 FINDHYPER 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.

## 4 Summarization of the covering database

In Sect. 3, we developed an efficient algorithm to find a set of hyperrectangles, *CDB*, to cover a transactional database. The summarization with false positives is more general than the summarization without a false positive (i.e. false positive ratio is 0). If there is no restriction on the false positive ratio, one can imagine that the summarization result is just one simple hyperrectangle $\mathcal{T} \times \mathcal{I}$ with cost $|\mathcal{T}| + |\mathcal{I}|$. Our empirical study also shows that summarization with false positives is generally more succinct (i.e. with less cost and less hyperrectangles) to reveal the high-level structure of the transactional database.

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 hyperrectangle $H' \in SCDB$, 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 in the following two questions:

1. Given the false positive budget $\beta$, $\frac{|SCDB^c \backslash DB|}{|DB|} \leq \beta$, 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 saving from merging $H_1$ and $H_2$ is

$$\begin{aligned} &cost(H_1) + cost(H_2) - cost(H_1 \oplus H_2) \\ &\quad = |T_i| + |T_j| + |I_i| + |I_j| - |T_i \cup T_j| - |I_i \cup I_j|. \end{aligned}$$

To minimize $cost(SCDB)$ with given false positive constraint $\frac{|SCDB^c \setminus DB|}{|DB|} \leq \beta$, we apply a greedy heuristic: *we will select two hyperrectangles in SCDB for combination 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 $H_j$,*

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

The numerator is the saving in $cost(SCDB)$ by merging $H_i$ and $H_j$, and the denominator is the number of false positives newly introduced. Therefore, our approach is to find a merge that yields highest saving in $cost(SCDB)$ per false positive quota. Algorithm 4 sketches the procedure which utilizes the heuristics.

**Algorithm 4** HYPER+($DB$, $CDB$, $\beta$)

1: $SCDB \leftarrow CDB$;
2: **while** $\frac{|SCDB^c \setminus DB|}{|DB|} \leq \beta$ **do**
3: find the two hyperrectangles $H_i$ 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** $SCDB$;

The second problem tries to merge the hyperrectangles in $CDB$ into $k$ 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* (Kellerer et al. 2004). However, since we consider only pair-wise merging, our algorithm does not have a guaranteed bound like the knapsack greedy algorithm. Algorithm 4 could be too time-costly when $|CDB|$ is large. In practice, we slightly revise Algorithm 4 and perform a random sampling merging to speed up the algorithm:

In each round, we randomly choose $C$ pairs of hyperrectangles among all possible pairs ($|SCDB|(|SCDB|-1)/2$) of hyperrectangles (when $C > |SCDB|(|SCDB|-1)/2$ we choose all). Then among the $C$ pairs of hyperrectangles we find two hyperrectangles $H_i$ and $H_j$ whose merge is within the false positive budget and produces the maximum saving-false positive ratio. Finally, we remove $H_i$ and $H_j$ from $SCDB$ and add $H_i \oplus H_j$ into $SCDB$. $C$ is an adjustable constant and the larger the $C$, the closer the random sampling merging algorithm to Algorithm 4, and when $C \geq |CDB|(|CDB|-1)/2$ the two algorithms are equal.

In Sect. 7, we show that the above greedy algorithm works effectively for both real and synthetic transactional datasets.

It is also worthwhile to mention that after merging hyperrectangles as discussed above, some existing hyperrectangles might become redundant, i.e. they are completely covered by some other hyperrectangles.

But it is expected that most redundant hyperrectangles are small when false positive budget is low (i.e. low merging activities). This is because most large hyperrectangles are expected to cover a good number of unique $DB$ elements as a consequence of HYPER algorithm, thus they are less likely to be completely covered after running HYPER+. It is a good idea to remove redundant hyperrectangles if the minimum $|SCDB|$ is desired. But it may not be worthwhile to remove a few small redundant hyperrectangles at the cost of identifying maximum number of redundant hyperrectangles, if the desired application primarily focuses on the first few large hyperrectangles discovered. In our empirical study we still keep the redundant hyperrectangles and leave the redundancy clearance problem as an open question to the readers.

## 5 Visualization

In many applications, people are not only interested in the patterns discovered from data, but also interested in how these patterns 'look'. Here, it is also quite natural for us to ask: "What is the best way to display those discovered hyperrectangles from a transactional database?"

In our visualization work (Jin et al. 2008), we refined the above question as: "Given a set of discovered hyperrectangles, how can we order the rows and columns of the transactional database to best display these hyperrectangles?" Furthermore, we formally defined the visualization cost and visualization problem as follows:

Given a database $DB$ with a set of hyperrectangles $CDB = \{H_1 = \{T_1 \times I_1\}, H_2 = \{T_2 \times I_2\}, \cdots, H_k = \{T_k \times I_k\}\}$, and two orders $\sigma_{\mathcal{T}}$ (the order of transactions) and $\sigma_{\mathcal{I}}$ (the order of items), the visualization cost of $CDB$, i.e., $visual\_cost(CDB, \sigma_T, \sigma_I)$, is

$$\sum_{j=1}^{k}(\max_{t_u \in T_j} \sigma_T(t_u) - \min_{t_w \in T_j} \sigma_T(t_w)) + \sum_{j=1}^{k}(\max_{i_u \in I_j} \sigma_I(i_u) - \min_{i_w \in I_j} \sigma_I(i_w)).$$

Given a database $DB$ with a set of hyperrectangles $CDB$, the visualization problem is to find the optimal orders $\sigma_T$ and $\sigma_I$, such that $visual\_cost(CDB, \sigma_T, \sigma_I)$ is minimized, i.e.,

$$\arg\min_{\sigma_T,\sigma_I} visual\_cost(CDB, \sigma_T, \sigma_I).$$

We answered the above question by linking the visualization problem to the well-known graph ordering problem, i.e., the minimal linear arrangement problem (Harper 1964; Safro et al. 2006). Interested readers may refer to Jin et al. (2008) for details of our hyperrectangle visualization algorithm. As a complement to our main results, we will show some visualization results in the experimental section.

## 6 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. Agrawal et al. (1998) define and develop a heuristic algorithm to represent a dense cluster in grid data using a set of rectangles. Further, Lakshmanan et al. (2002) consider the situation where false positives are allowed. Recently, Gao and Ester (2006) extend descriptive data mining from a clustering description to a discriminative setting using a rectangle notion. 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 hyperrectangle. 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, which is potentially more precise and 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 sets (Wang and Karypis 2006). 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 using a single-transaction hyperrectangle from $\{\{t\} \times I_i : I_i \in \mathcal{F}_\alpha, t \in T(I_i)\}$ to maximally cover each transaction. Chandola and Kumar compress datasets of transactions with categorical attributes into informative representations by summarizing transactions (Chandola and Kumar 2007). 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 transaction-item pairs, 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. (2006), van Leeuwen et al. (2006), Vreeken et al. (2007). In Siebes et al. (2006), van Leeuwen et al. (2006), 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 hyperrectangles 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 (Vreeken et al. 2007). 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 hyperrectangles, and their cost is based on the MDL principle. In addition, we also study how the hyperrectangle can be further summarized by allowing false positives. Thus, our methods can provide a much more succinct summarization of the transactional database. Finally, we not only provide rigorous proofs on the hardness of various summarization problems, but also develop a polynomial-time algorithm with a proven approximation bound.

**Co-clustering and Numerical Data Mining:** As mentioned before, co-clustering (Hartigan 1972; Mirkin 1996) 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 (Li 2005). 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 positives 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).

In Besson et al. (2006), the authors try to identify relevant pattern (bi-sets), which is defined similar to hyperrectangle in this paper, to mine numerical data. A bi-sets pattern is a Cartesian product between $X$ rows and $Y$ columns, in which each element has different numerical values. It is required that the difference between the maximum value and the minimum value in a bi-sets pattern is no larger than a threshold $\epsilon$. Although the definition of bi-sets and hyperrectangle are related, their goals and their applications are quite different: By defining bi-sets, the authors try to identify relevant numerical patterns in numerical data, while our goal is to use hyperrectangles to summarize 0/1 transactional databases.

**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. First, it tries to recognize the frequent itemsets when some noise is added into the data. In other words, the frequent itemsets are disguised in the data. Second, it provides a way to reduce the number of frequent itemsets since many of these frequent itemsets can be recognized as the variants of a true frequent itemset. This in general is referred to as pattern summarization (Afrati et al. 2004; Pei et al. 2004). Most of the efforts in error-tolerant frequent itemsets can be viewed as finding dense hyperrectangles with certain constraints. The support envelope notion proposed by Steinbach et al. (2004) 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, Johnson et al. (2004), 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, which aims to emphasize the important characteristics of the data.

**Set Covering:** From the theoretical computer science viewpoint, our problem can be generalized as a variation of the *set covering* problem. Similar to the problem studied in Gao et al. (2007), our covering problem does not directly have a list of candidate sets as in traditional set covering, because our total set of candidate sets is too large to be materialized. The problem and solution studied in Gao et al. (2007) cannot be applied to our problem as it tries to find a minimal number of sets for covering. The strategy proposed in this work to handle this variation of the set covering problem is also very different from Gao et al. (2007). In Gao et al. (2007), the strategy is to transform the set cover problem into an independent vertex set problem. The graph in the new problem space contains all the elements in the universal (or grounding) set which needs to be covered. Any two elements in the grounding set can potentially be put into one candidate set for covering if connected with an edge. Then, finding a minimal set cover is linked to finding an independent vertex set, and a heuristic algorithm progressively collapses the graph to identify the set cover. Considering the number of elements in the transaction database, this strategy is too expensive and the solution is not scalable. Here, we propose to identify a large family of candidate sets which is significantly smaller than the number of all candidate sets but is deemed sufficient for set covering. Then, we investigate how to efficiently process these candidate sets to find an optimal set cover.

**Tiling Databases:** The concept of a tile, which is very similar to a hyperrectangle, has been defined previously in Geerts et al. (2004); Gionis et al. (2004) to cover transactional databases for knowledge discovery purposes. The tile concepts defined in Geerts et al. (2004) and Gionis et al. (2004) are slightly different.

Geerts et al. (2004), define a tile corresponding to an itemset to be all transaction-item pairs resulting from the Cartesian product between the itemset and its supporting-transaction set.

Gionis et al. (2004) define a basic tile to be a Cartesian product between a set of rows and a set of columns, and the Cartesian product is associated with a probability $p$. They also define a hierarchical tile to be a basic tile plus a set of disjoint exception tiles.

In our hyperrectangle definition, we only require a Cartesian product be between a itemset and a subset of its supporting-transaction when no false positive is allowed. When false positives are allowed, a Cartesian product can be between any number of rows (i.e. transactions) and any number of columns (i.e. items). Therefore our hyperrectangle definition is more general than the tile definition in Geerts et al. (2004), but very close to the basic tile definition in Gionis et al. (2004). The only difference between our hyperrectangle definition and the basic tile definition in Gionis et al. (2004) is that our definition does not associate a probability to each Cartesian product.

**Table 1** Dataset characteristics

| Datasets | $\mathcal{I}$ | $\mathcal{T}$ | Avg. Len. | $\|DB\|$ | Density |
|---|---|---|---|---|---|
| Chess | 75 | 3,196 | 37 | 118,252 | Very 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 | Very sparse |

Based on our simple definition of a hyperrectangle, we propose a complete set of theorems and algorithms for efficient transactional database summarization.

**Itemsets other than frequent:** Frequent itemset mining and the concept of frequent itemsets (Agrawal et al. 1993) are well known by now. Since the set of frequent itemsets is very important in succinctly describing a transactional database, we use it as an input for our HYPER algorithm. By adjusting the support level $\alpha$, we can let the set of frequent itemsets include all itemsets or most frequent itemsets.

However, frequent itemsets themselves may not be succinct enough for transactional data mining purposes. The well-known closed frequent itemsets and maximal frequent itemsets (Han and Kamber 2006) give concise presentations of frequent itemsets. Recently, free-sets by Boulicaut et al. (2003), $\delta$-cluster by Xin et al. (2005), nonderivable itemset by Calders and Goethals (2007), and Cartesian contour by Jin et al. (2009) are defined and can be used to further summarize frequent items. All these concepts aim at finding most essential information from huge amounts of itemsets.

Although in our algorithm HYPER we use the set of frequent itemsets as a default input, the set of frequent itemsets can be replaced by any set of itemsets. Therefore HYPER can serve as a platform for inputting any summarizing itemsets to summarize a transactional database. In the experimental section we also report the summarization results by replacing the set of frequent itemsets with the set of frequent closed itemsets and also the set of maximal frequent itemsets.

## 7 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. We use MAFIA[2] Burdick et al. (2005)[3], a very good open source software for quickly generating frequent itemset (*fi*), maximal frequent itemset (*mfi*), and frequent closed itemset (*fci*).

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

[2] In Xiang et al. (2008), the conference version of this paper, we used Borgelt's Apriori software (http://fuzzy.cs.Uni-Magdeburg.de/%7Eborgelt/software) under its default setting (version 4.08–4.31), which generates a frequent itemset with at most 5 items. In this experimental study, we use MAFIA under its default setting, which has no limit on the size of an itemset. Therefore, values obtained in this experimental study are different from the conference version.

[3] Software available online: http://himalaya-tools.sourceforge.net/Mafia/.

Our algorithms were implemented in C++ and run on Linux 2.6 on an AMD Opteron 2.2 GHz with 4GB of memory.

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

1. How well can HYPER (Algorithm 3) and HYPER+ (Algorithm 4) summarize a transactional dataset with respect to the summarization cost?
2. How well can the false positive condition improve the summarization cost?
3. How do the sets of frequent itemsets, maximal frequent itemsets, and frequent closed itemsets at different minimum support levels ($\alpha$) 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.

### 7.1 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 Figs. 4a, b, 5a, b, we show the summarization cost with respect to different support levels $\alpha$ on the chess, mushroom, pumsb_star and T10I4D100K datasets. Each of these four figures has a total of 8 lines. Two of them are *reference lines*: the first reference line, named "DB", is the value of $|DB|$, i.e. the number of elements in $DB$. Recall that in the problem formulation, we denote $cost(CDB_H) = |\mathcal{T}| + |DB|$ and $cost(CDB_V) = |\mathcal{I}| + |DB|$. Thus, the upper bound of cost is $|DB| + \min\{|\mathcal{T}|, |\mathcal{I}|\}$. Since $|DB| \geq \max\{|\mathcal{T}|, |\mathcal{I}|\}$, This reference line $|DB|$, is no less than half of 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 "fi_CDB" line records the cost of $CDB$ being produced by HYPER. The "fi_SCDB_0.1", "fi_SCDB_0.2", and "fi_SCDB_0.4" lines record the cost of $SCDB$ being produced by HYPER+ with 10%, 20%, and 40% false positive budget. The "mfi_CDB" line records the cost of $CDB$ being produced by HYPER with frequent itemsets being replaced by maximal frequent itemsets. The "fci_CDB" line records the cost of $CDB$ being produced by HYPER+ with frequent itemsets being replaced by frequent closed itemsets.

Accordingly, in Figs. 4c, d, 5c, d, we show the number of hyperrectangles (i.e. $k$) in the covering database $CDB$ or $SCDB$ at different support levels. The "fi_CDB_k" line records $|CDB|$, and the "fi_SCDB_0.1_k", "fi_SCDB_0.2_k", "fi_SCDB_0.4_k" lines record $|SCDB|$ being generated by HYPER+ with 10%, 20%, and 40% false positive budget. The "mfi_CDB_k" line records $|CDB|$ being generated by HYPER with frequent itemsets being replaced by maximal frequent itemsets. The "fci_CDB_k" line records $|CDB|$ being generated by HYPER with frequent itemsets being replaced by frequent closed itemsets.

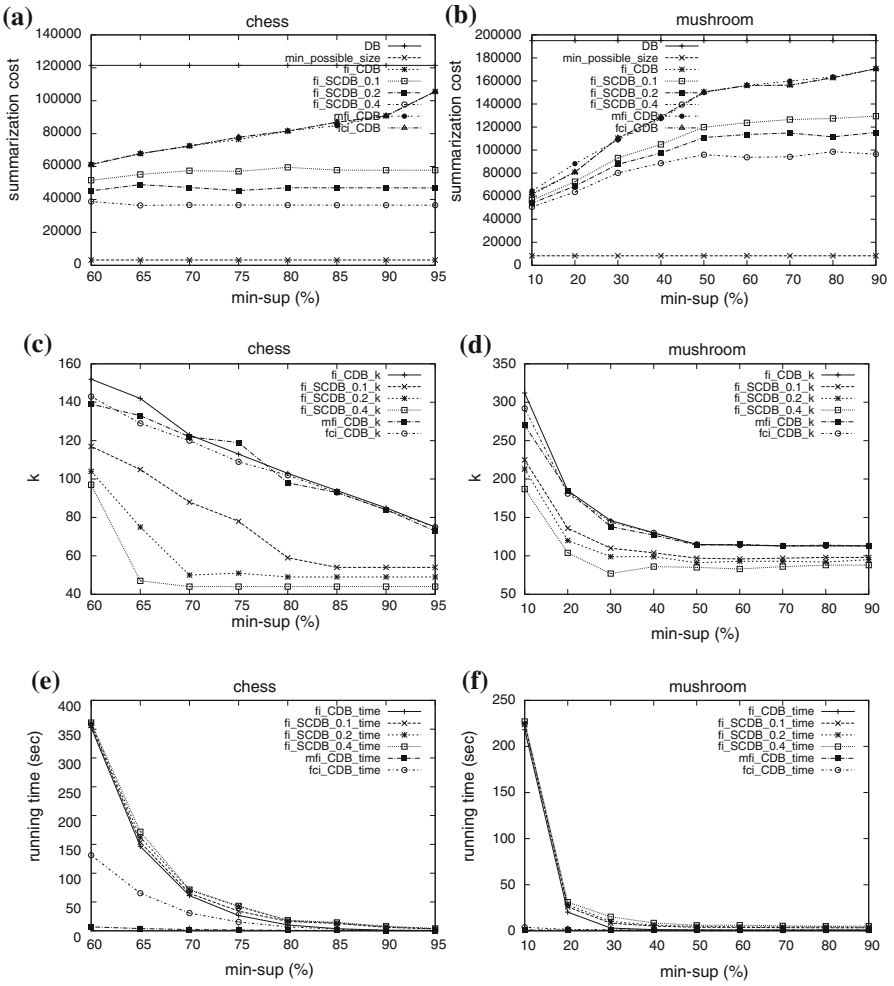



**Fig. 4** Experimental results. **a** Chess summarization cost varying $\alpha$. **b** Mushroom summarization cost varying $\alpha$. **c** Chess hyperrectangle number varying $\alpha$. **d** Mushroom hyperrectangle number varying $\alpha$. **e** Chess running time varying $\alpha$. **f** Mushroom running time varying $\alpha$

Figures 4e, f, 5e, f show the running time. Here the line "fi_CDB_time" records the running time of HYPER generating *CDB* from *DB* given frequent itemsets of *DB*. The "mfi_CDB_time" line records the running time of HYPER generating *CDB* from *DB* given maximal frequent itemsets of *DB*. The "fci_CDB_time" line records the running time of HYPER generating *CDB* from *DB* given frequent closed itemsets of *DB*.

The "fi_SCDB-0.1_time", "fi_SCDB-0.2_time", and "fi_SCDB-0.4_time" 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 MAFIA that is being used to generate those itemsets.

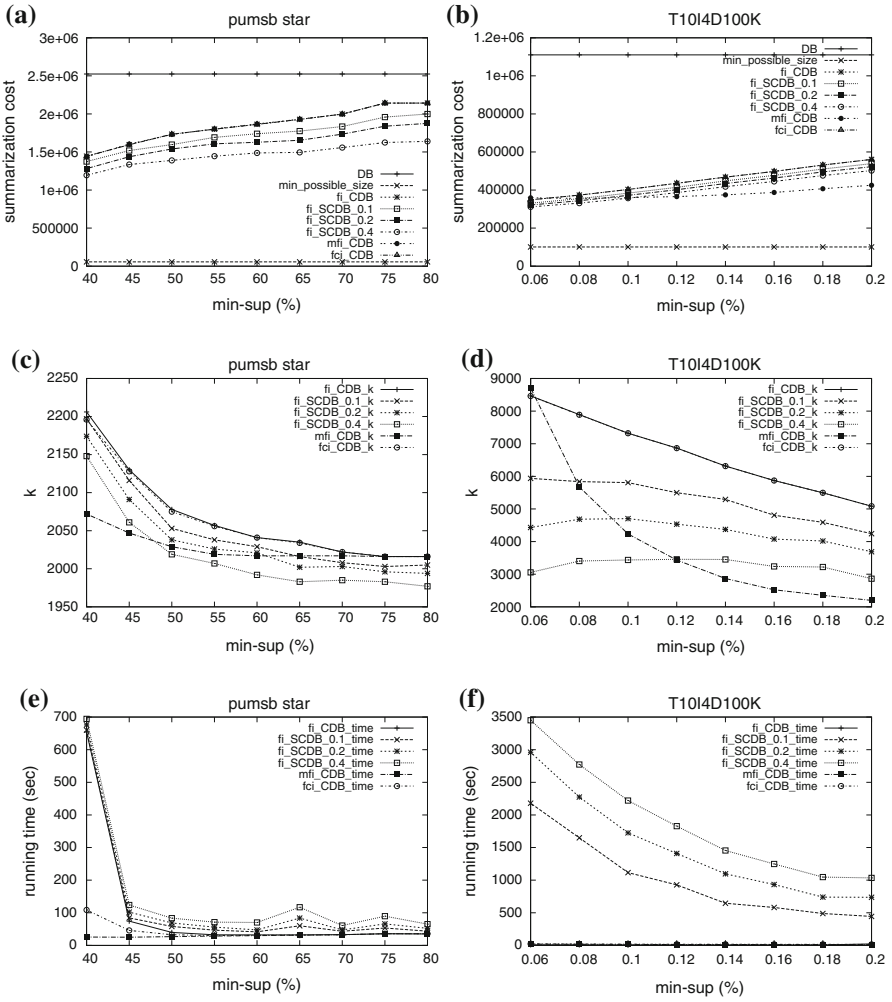



**Fig. 5** Experimental results. **a** pumsb_star summarization cost varying $\alpha$. **b** T10I4D100K summarization cost varying $\alpha$. **c** pumsb_star hyperrectangle number varying $\alpha$. **d** T10I4D100K hyperrectangle number varying $\alpha$. **e** pumsb star running time varying $\alpha$. **f** T10I4D100K running time varying $\alpha$

Here, we can make the following observations:

1. The summarization cost reduces as the support level $\alpha$ 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 ($\mathcal{C}_\alpha^-$) 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. 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 (excluding mfi_CDB_time and fci_CDB_time) on all these datasets. First, we observe the summarization cost tends to converge when $\alpha$ drops. Second, we can see that for mushroom and pumsb_star, the number of hyperrectangles $k$ tend to increase fast when $\alpha$ drops below some threshold, and the running time, which shares the same threshold, increases accordingly. (This phenomenon is less obvious in very sparse dataset T10I4D100K and very dense dataset chess, which have relatively less number of optimal covering choices.) However, the convergence behavior tends to maintain the summarization cost at the same level or only decrease it slightly. This we believe suggests that a lot of smaller hyperrectangles are chosen without reducing the cost significantly, and that these small hyperrectangles are of little benefit to the data summarization. This phenomenon suggests that a reasonably high $\alpha$ can produce a comparable summarization as a low $\alpha$ with much less computational cost, which would be especially important for summarizing very large datasets.

3. Frequent closed itemsets and maximal frequent itemsets are good substitutes for frequent itemsets in HYPER and HYPER+. We can see in most experiments, the summarization cost of HYPER or HYPER+ with maximal frequent itemsets or frequent closed itemsets, are very close to the results with frequent itemsets, but the running time is much shorter under small supports. This suggests that frequent closed itemsets and maximal frequent itemsets are good approximations for frequent itemsets. Recall that under the same support level, $mfi \subseteq fci \subseteq fi$, and when the support level is very low, $|fi|$ is much larger than $|fci|$ and $|mfi|$. This also explains why for each dataset, mfi_CDB_time and fci_CDB_time do not have the similar behaviors as other running times.

### 7.2 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 Figs. 6a–f, each method (i.e. HYPER+ with *fi*, *fci*, and *mfi*) has two lines which correspond to two different minimum support levels $\alpha$ for generating *SCDB*. For instance, support_15_fi is the 15% minimal support for the HYPER+ using frequent itemsets.

Here in Fig. 6a, c, e, 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, in Fig. 6b, d, f, 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, tend 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.,

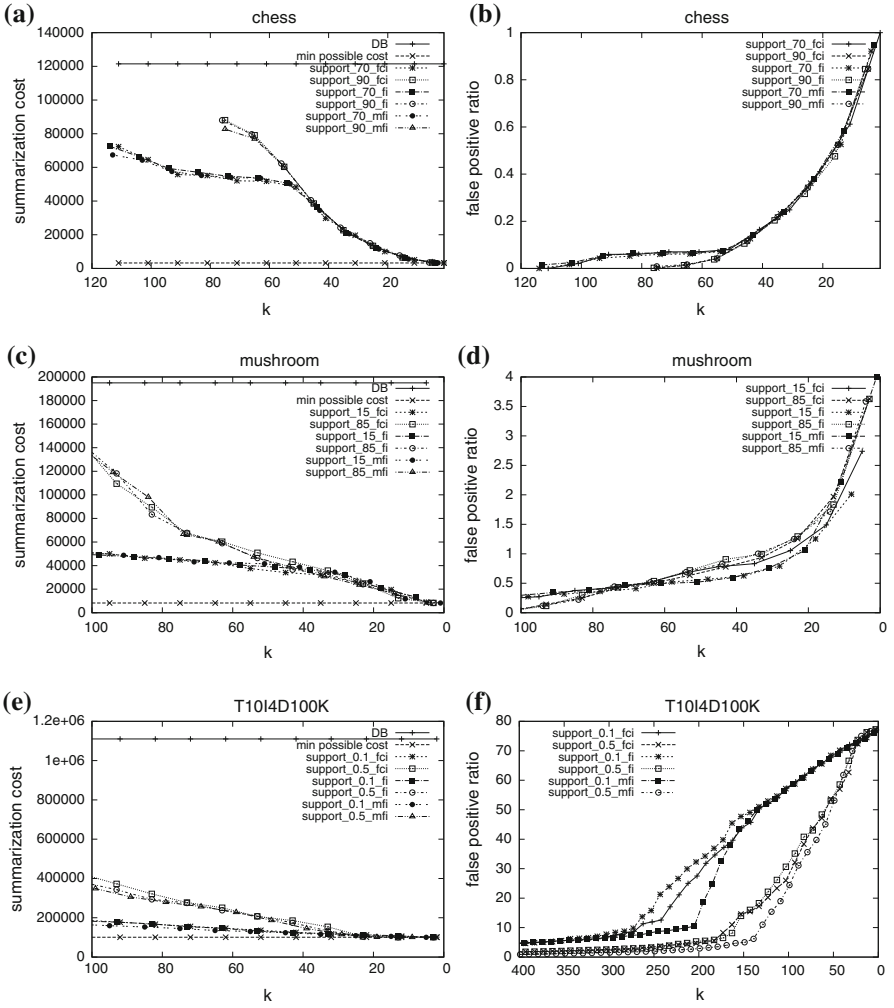



**Fig. 6** Experimental results. **a** Chess cost varying $k$. **b** Chess false positive ratio varying $k$. **c** Mushroom cost varying $k$. **d** Mushroom false positive ratio varying $k$. **e** T10I4D100K cost varying $k$. **f** T10I4D100K false positive ratio varying $k$. $k$ denotes the number of hyperrectangles

there is only around 1% ones in the binary matrix. Using the minimal support 0.5% and $k = 200$ and maximal frequent itemsets, our false positive ratio is around 300%, which suggests that we use around 4% of the cells in the binary matrix to summarize T10I4D100K.

## 7.3 Hyperrectangle visualization

As a complement to our main results, in this subsection we show some visualization results of discovered hyperrectangles.

In Fig. 7 we display visualization effects of hyperrectangles on sampled datasets of "mushroom" and "T10I4D100K" by our visualization algorithm (Jin et al. 2008). The visualization method can be incorporated into an interactive visualization environment to allow users to quickly have some intuitive ideas of the data and its main patterns, such as how dense the data is, how large the main patterns are, etc.

To illustrate how hyperrectangles are visualized, we first select a number of hyperrectangles from a transactional database for our visualization purpose. Typically we select the first several hyperrectangles outputted by HYPER or HYPER+. Then we run our visualization algorithm to determine the order of transactions and the order of items. Recall the visualization cost in Sect. 5. Intuitive speaking, our visualization algorithm tries to find an order of transactions and an order of items such that for most hyperrectangles, their cells are getting closer in 2-dimension. We mark a pixel $(i, j)$ black if the transaction at order $i$ contains the item at order $j$, otherwise white. For each hyperrectangle, we draw a minimum bounding box - the smallest rectangle in 2-dimension that covers all the cells of the hyperrectangle. Hence, one can also conclude that our visualization algorithm aims at minimizing the total perimeters of all bounding boxes. In some cases all the cells inside the bounding box belong to the corresponding hyperrectangle and the area inside the bounding box is completely black. But in many cases the bounding rectangle may contains white pixels, even if false positive is prohibited. This is because given an order of transactions and an order of items, there may exist hyperrectangles whose cells are scattered on the 2-dimension.

We selected two datasets, mushroom (dense) and T10I4D100k (sparse), for our visualization illustration. To let the visualization best fit the limited space on the paper, we choose partial data from these datasets. Specifically, we randomly sampled 1000 transactions from each datasets, so that the number of transactions are more in line with the number of items. Then we apply HYPER on each sampled dataset (frequent itemsets with 10% support for the sampled mushroom dataset, and frequent itemsets with 0.5% support for the sampled T10I4D100k dataset) and collect the first 10 outputted hyperrectangles for our visualization. We display four figures for each sampled dataset: Figs. 7a and 8a show their appearances with original transaction order ($\sigma_T$) and item order ($\sigma_I$). Figures 7b and 8b show their appearances with updated orders by our visualization algorithms for the best visualization of the ten hyperrectangles. In Figs. 7c and 8c, we highlight the first five hyperrectangles by zooming in and drawing a colored bounding box around each hyperrectangle. We highlight the second five hyperrectangles in Figs. 7d and 8d in the same way as we do in Figs. 7c, 8c. Interested readers may refer to Jin et al. (2008) for more visualization results.

### 7.4 Case study on real datasets

In this subsection, we give a case study to show how our methods are useful in real data mining applications, we test them on real datasets.

We built two datasets of KDD author / keyword correspondence. From the DBLP repository, we extracted a list of KDD papers, and augmented each paper with abstract text extracted from the ACM Digital Library, where possible. We then took the 972 KDD papers with complete information, processed their abstract and keyword text

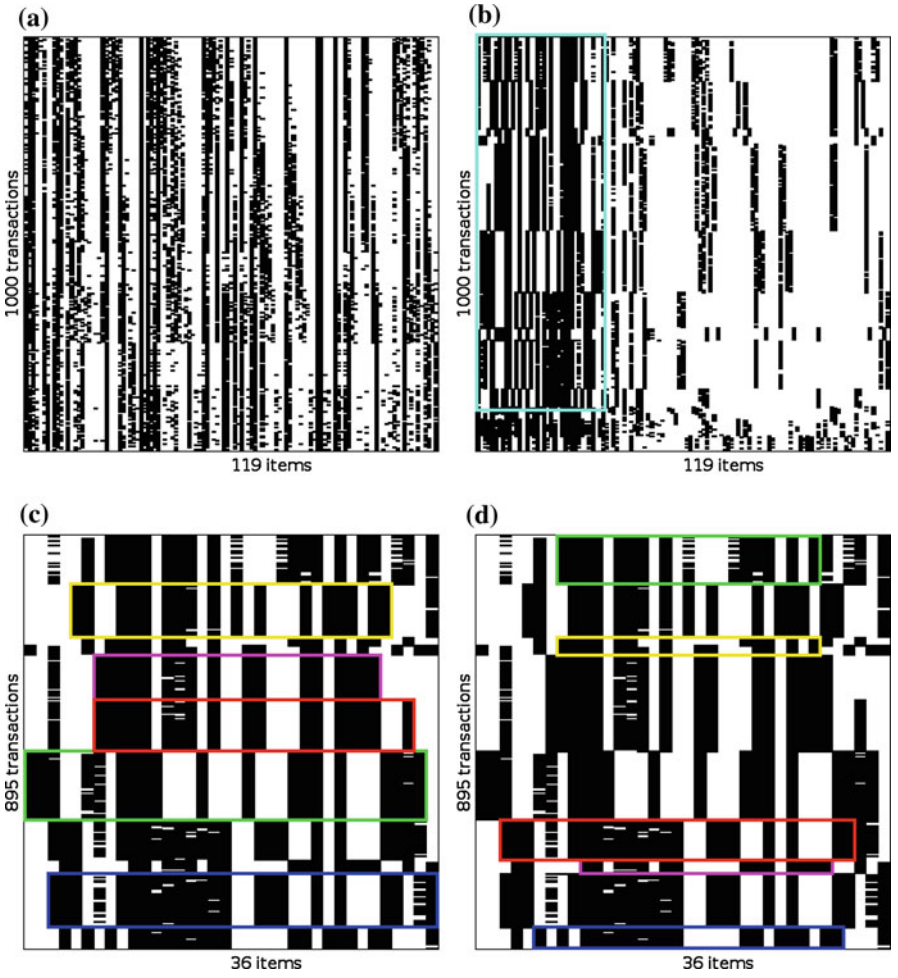



**Fig. 7** Visualization results (hyperrectangles best viewed in color). **a** Sampled mushroom in original orders. **b** Sampled mushroom in new orders. **c** Hyperrectangles 1–5 of sampled mushroom after zooming in **d** Hyperrectangles 6–10 of sampled mushroom after zooming in. (Color figure online)

with full text search tools (from a standard database engine) to stem them and remove stop words, and chose 114 common and representative words to serve as the feature vector for all papers. Common words are words (except prepositions and articles) appearing in high frequency. Representative words, which are selected subjectively by us, are in our opinion good representatives of research topics. The larger dataset *KDD* contains all author / keyword correspondences. The smaller dataset *KDD2008* contains only author / keyword correspondences made from KDD 2008 papers. In these datasets, each word is represented as an item, and each author is represented as a transaction.

By applying HYPER and HYPER+ algorithms, we can get hyperrectangles outputs as summarizations for the above datasets. For example, we apply HYPER+ on both *KDD* and *KDD2008* with 1% support and a 10% false positive ratio. To

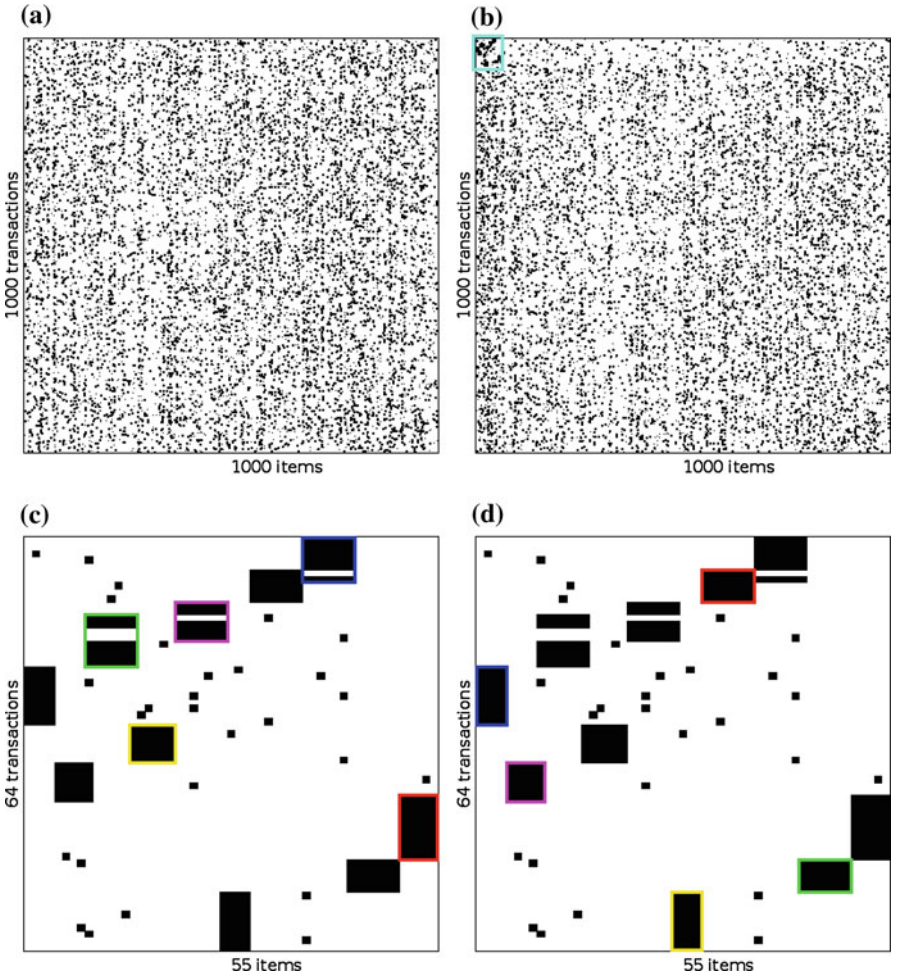



**Fig. 8** Visualization results. **a** Sampled T10I4D100K in original orders. bf b Sampled T10I4D100K in new orders. **c** Hyperrectangles 1–5 of sampled T10I4D100K after zooming in. **d** Hyperrectangles 6–10 of sampled T10I4D100K after zooming in

facilitate the following discussion, let us name the results as *KDD*_0.01_0.1 and *KDD*08_0.01_0.1, respectively. We get around 400 hyperrectangles in *KDD*_0.01_0.1 for covering the dataset *KDD*, and around 130 hyperrectangles in *KDD*08_0.01_0.1 for covering the dataset *KDD2008*. Among these covering hyperrectangles, about 30% or less contain at least 4 keywords and at least 8 authors. These hyperrectangles may be most informative to readers. This also suggests that we may use a relatively small amount of hyperrectangles to summarize the major characters of a dataset.

In outputs by HYPER and HYPER+, we can observe which set of authors are related to which set of keywords. This result is interesting to many authors because they can find who else are working on the similar topics. For example, in *KDD*08_0.01_0.1, we discover a hyperrectangle as follows: (Note: Keywords are stems.)

*Authors*: A Sridharan, C Faloutsos, D Cosley, DB Neill, DP Huttenlocher, DJ Crandall, J Bolot, JG Schneider, JM Kleinberg, J Leskovec, K Das, L Backstrom, M Seshadri, S Suri, S Machiraju
*Keywords*: network, social, massiv, behavior, process, time;

It shows many authors of KDD08 who are related to social network research with emphasis on its behavior, process and time.

When false positive ratio increases, in the summarization results, authors are more likely to be associated with some topics (i.e., keywords) they rarely (or never) worked on. This result might be interesting to some authors because it provides an interesting indication on the possible extensions or related work to their research.

As an example of facilitating the analysis of the major summarization results for dataset *KDD* and *KDD2008*, we visualize the first 10 hyperrectangles from $KDD\_0.01\_0.1$ in Fig. 9c, e, and the first 10 hyperrectangles from $KDD08\_0.01\_0.1$ *KDD2008* in Fig. 9d, f. Recall that hyperrectangles are not necessarily rectangles, and the general objective of our visualization is to bring hyperrectangles as close as possible in 2-dimension. We draw a bounding rectangle for each hyperrectangle, which is the smallest rectangle that covers all of its cells. (However, it does not mean that all cells inside the bounding box belong to this hyperrectangle.) Incorporating our visualization algorithm into an interactive visualization platform, we can imagine that by clicking a bounding rectangle, one can see the actual authors-keywords relationship.

From this visualization, one can tell roughly how large or how dispersed a hyperrectangle could be. More importantly, we can observe the interaction between hyperrectangles. For example, the magenta bounding box is completely enclosed by the red bounding box in Fig. 9c. This suggests it is very likely that hyperrectangle 5 is somehow related to hyperrectangle 1. By examining the actual authors and keywords included in these two hyperrectangles (see below), we conclude that our guess is legitimate.

Hyperrectangle 1 in Fig. 9c:
*Authors*: AW Moore, CC Aggarwal, C Faloutsos, D Fuhry, D Xin, F Mrchen, FF Dragan, H Mannila, H Xiong, JX Yu, J Pei, J Wang, J Han, J Yang, M Ester, M Hu, M Vlachos, MJ Zaki, PN Tan, PS Yu, R Jin, S Jaroszewicz, V Kumar, W Su, W Wang, W Jin, X Yan, X Wu, Y Xiang
*Keywords*: synthet, databas, frequent, itemset, prune, process, mine, cluster, time, algorithm, pattern
Hyperrectangle 5 in Fig. 9c:
*Authors*: A Ghoting, AKH Tung, B Field, BJ Gao, C Wang, D Polshakov, E Stroulia, G Agrawal, G Fang, G Cong, G Buehrer, H Saigo, H Xiong, K Tsuda, LH Yang, M Steinbach, M El-Ramly, N Krmer, P Smyth, PG Sorenson, R Gupta, R Kohavi, S Jaroszewicz, W Peng, W Wang, X Guo
*Keywords*: frequent, process, mine, algorithm, pattern

Complemented by another important hyperrectangle found in $KDD\_0.01\_0.1$, as listed in the following, we find important authors and topics in frequent itemset mining of KDD conferences.

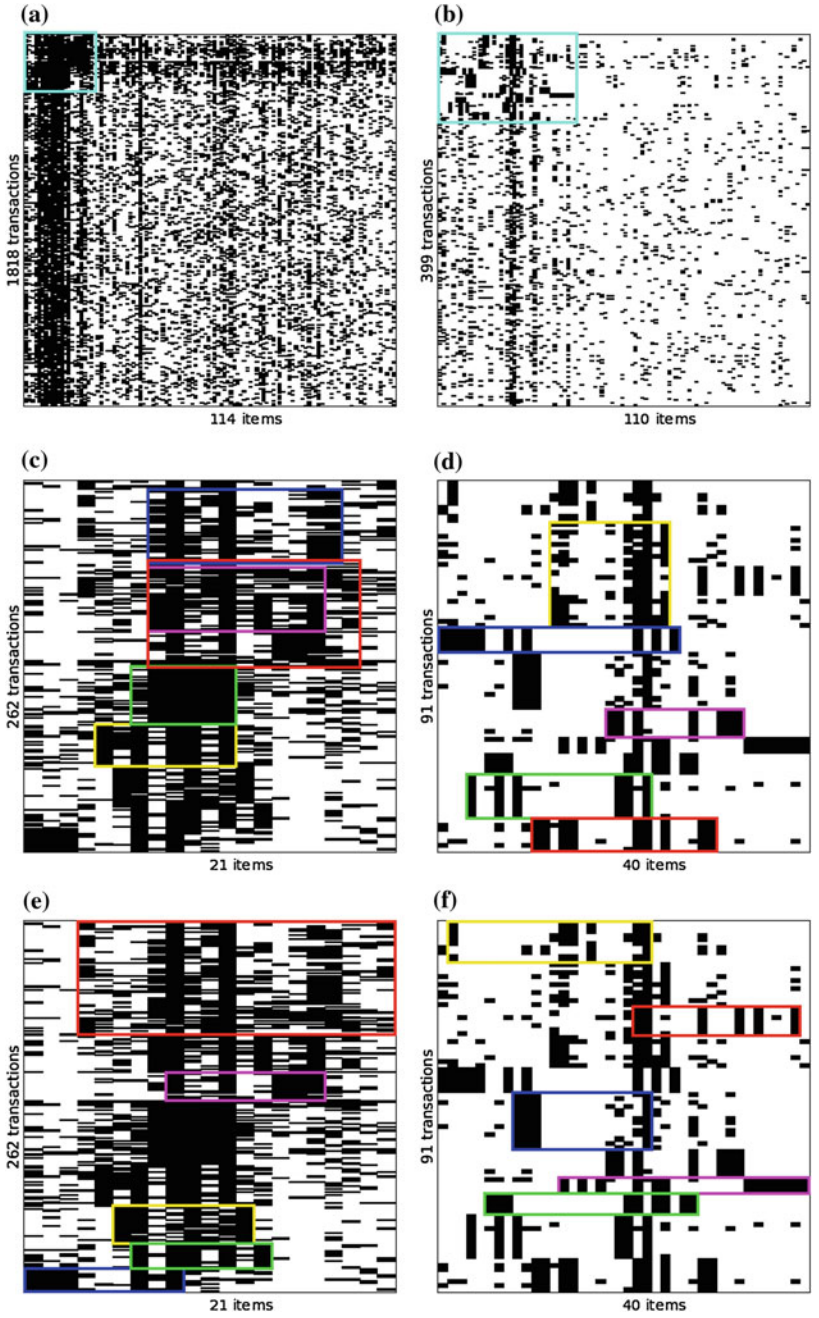



**Fig. 9** KDD author keyword visualization. *Red*(1,6), *green*(2,7), *blue*(3,8), *yellow*(4,9), *magenta*(5,10). *Red*(1,6) means hyperrectangle 1 and 6 are visualized by *red* bounding boxes. Others can be explained in the same way. **a** *KDD* in new orders. **b** *KDD*2008 in new orders. **c** Hyperrectangles 1–5 of *KDD* after zooming in. **d** Hyperrectangles 1–5 of *KDD*2008 after zooming in. **e** Hyperrectangles 6–10 of *KDD* after zooming in. **f** Hyperrectangles 6–10 of *KDD*2008 after zooming in. (Color figure online)

*Authors*: A Bifet, A Grama, B Shi, B Mobasher, B Goethals, C Liu, C Wang, D Lo, F Chen, F Pan, FSL Brinkman, H Wang, J Prins, JL Gardy, J Cai, J Li, J Huan, K Sinha, K Zhang, L Wong, LD Raedt, M Koyutrk, M Hu, O Schulte, O Verscheure, R Gavald, R She, RW White, SC Khoo, T Calders, V Tankasali, W Su, X Shen, Y Zhu
*Keywords*: frequent, mine, algorithm, pattern

### 7.5 Empirical comparison with related work

In this section, we compare HYPER with related work through experiments. Although there are no universal standards on what a good summarization scheme is, we believe the following comparison on the effectiveness of summarization gives readers some ideas on how HYPER compares to related work.

The following two factors, in our opinion, are good descriptions of the effectiveness of summarization methods:

- How well do the major summarization results capture the nature of the data?
- How concise are these summarization results?

Given the above two factors, we are primarily interested in studying the first $k$ hyperrectangles discovered by HYPER algorithm according to the following two criteria:

- The total number of elements in *DB* (i.e., cells that are 1 in the original matrix) the first $k$ hyperrectangles cover.
- The ratio of coverage per cost, i.e., the total number of covered elements over the total cost of the $k$ hyperrectangles. (See Sect. 1.1 for the definition of cost.)

We compare HYPER with tiling (Geerts et al. 2004) and NDI Calders and Goethals (2007). Both tiling and NDI only output itemsets. We build hyperrectangles for them by associating supporting transactions with each itemset.

The source code of tiling is provided by the authors and implemented in Java. The source of NDI is made available publicly by the authors. Our PC platform (for running Java program of tiling) is an AMD Opteron 2.2 GHz with 4 GB of memory.

Since the tiling program implemented in Java cannot run efficiently on large inputs on our PC settings, we test all three methods on two sampled datasets: The sampled mushroom dataset consisting of 250 randomly selected transactions from the dense dataset mushroom, and the sampled pumsb_star dataset consisting of 250 randomly selected transactions from the sparse dataset pumsb_star.

For HYPER, we not only use the default frequent itemsets (i.e., *fi*), but also use the frequent closed itemsets (i.e., *fci*) as inputs. As we have discussed in Sect. 7.1, HYPER can take a lower support-level on *fci* than *fi* given the same running time. Hence, we test HYPER on the sampled mushroom dataset with frequent itemsets of 7% support-level (fi-7), and frequent closed itemsets of 1% support-level (fci-1). We also test HYPER on the sampled pumsb_star dataset with frequent itemsets of 30% support-level (fi-30), and frequent closed itemsets of 4% support-level (fci-4).

To be comparable to HYPER, we tried the same support levels on NDI (ndi-7 and ndi-1 for the sampled mushroom; ndi-30 and ndi-4 for the sampled pumsb_star). Since NDI does not rank the outputs, we need to select the $k$ hyperrectangles. To be as fair

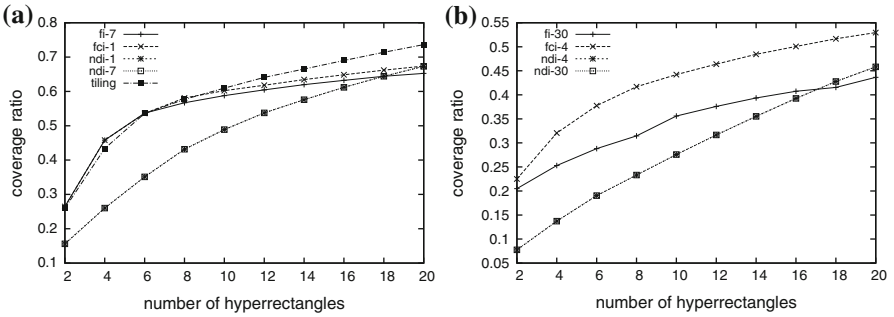



**Fig. 10** Coverage on sampled datasets. **a** Coverage on sampled mushroom. **b** Coverage on sampled pumsb_star

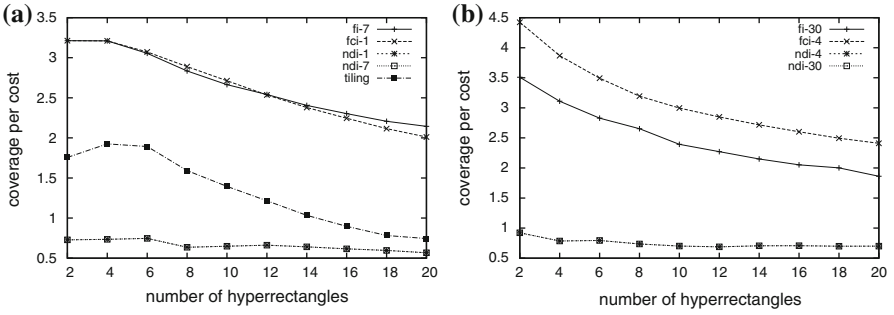



**Fig. 11** Coverage per cost on sampled datasets. **a** Coverage per cost on sampled mushroom. **b** Coverage per cost on sampled pumsb_star

as we can, we select hyperrectangles from NDI outputs according to the greedy set cover approach, i.e., in each iteration select the hyperrectangle that can cover the most uncovered elements.

The tiling program for mining maximum k-tilings does not take support-levels as inputs. Since it exactly output $k$ tiles, we do not need to make additional selections.

Here are the results of comparison:

We vary $k$ from 2 to 20. The total coverage (displayed as the ratio of coverage over the total elements) on each sampled dataset by the $k$ hyperrectangles is shown in Fig. 10, the ratio of coverage per cost on each sampled dataset is shown in Fig. 11. The Java-based tiling program cannot output a result on the sampled pumsb_star in a reasonable time therefore it is missing in Figs. 10b and 11b. We do not report the running time for our empirical comparison due to the platform disparateness (i.e., tiling in Java but NDI and HYPER in C++).

From Fig. 10, we can see that when $k$ increasing, the coverage ratio of all the methods increases, as expected. Generally speaking, the coverage ratios of these methods do not differ much. However, tiling slightly outperform HYPER on the sampled mushroom dataset, and HYPER is slightly better than NDI in most of the cases. This is understandable because the goal of tiling is directly towards maximum coverage, while the goal of NDI is to find a minimal representation for all frequent itemsets with respect to the deduction rules. For NDI, the coverage ratio is almost collinear with the number of

hyperrectangles. This suggests that largest $k$ hyperrectangles constructed from non-derivable itemsets have very limited overlapping, if any, and their sizes (Note $|H|$ is the size of hyperrectangle $H$) are comparable. In contrast, the coverage ratio of tiling or HYPER increases quickly at the beginning but slowly at the end when $k$ increases, which suggests that both methods are quite successful in identifying largest hyperrectangles.

Although tiling and HYPER have similar performance in coverage, HYPER significantly outperforms tiling in coverage per cost, as shown in Fig. 11. This confirms that HYPER achieves its goal in succinctly summarizing transactional datasets. The decreasing of coverage per cost in HYPER and in tiling justify the motivation of finding major hyperrectangles for succinctly summarizing transactional datasets. The almost flat lines of NDI in Fig. 11 again suggest that the largest $k$ hyperrectangles constructed from non-derivable itemsets have similar sizes but dissimilar coverages.

## 8 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 transactional databases. 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 addition, we showed how to visualize hyperrectangles to facilitate the analysis of the summarization results. In the future, we would like to investigate those behaviors analytically and thus produce better summarizations. We also plan to apply this method on real world applications, such as biomedical data analysis, for which we conjecture some hyperrectangles in biomedical data may correspond to certain biomarkers.

**Acknowledgment** This work was supported in part by the National Science Foundation under Grant #1019343 to the Computing Research Association for the CIFellows Project.

## References

Afrati FN, Gionis A, Mannila H (2004) Approximating a collection of frequent sets. In: KDD, pp 12–19

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: VLDB, pp 487–499

Agrawal R, Borgida A, Jagadish HV (1989) Efficient management of transitive relationships in large data, knowledge bases. In: SIGMOD Conference, pp 253–262

Agrawal R, Imielinski T, Swami AN (1993) Mining association rules between sets of items in large databases. In: SIGMOD Conference, pp 207–216

Agrawal A, Mannila H, Srikant R, Toivonen H, Verkamo A (1996) Fast discovery of association rules. Adv Knowl Discov Data Min 307–308

Agrawal R, Gehrke J, Gunopulos D, Raghavan P (1998) Automatic subspace clustering of high dimensional data for data mining applications. In: SIGMOD Conference, pp 94–105

Besson J, Robardet C, De Raedt L, Boulicaut J-F (2006) Mining bi-sets in numerical data. In: KDID, pp 11–23

Boulicaut J-F, Bykowski A, Rigotti C (2003) Free-sets: a condensed representation of boolean data for the approximation of frequency queries. Data Min Knowl Discov 7(1):5–22
Burdick D, Calimlim M, Flannick J, Gehrke J, Yiu T (2005) Mafia: a maximal frequent itemset algorithm. IEEE Trans Knowl Data Eng 17(11):1490–1504
Calders T, Goethals B (2007) Non-derivable itemset mining. Data Min Knowl Discov 14(1):171–206
Chandola V, Kumar V (2007) Summarization—compressing data into an informative representation. Knowl Inf Syst 12(3):355–378
Chvátal V. (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4:233–235
Faloutsos C, Megalooikonomou V (2007) On data mining, compression, kolmogorov complexity. Data Min Knowl Discov 15(1):3–20
Gao Byron J, Ester M (2006) Turning clusters into patterns: rectangle-based discriminative data description. In: ICDM, pp 200–211
Gao Byron J, Ester M, Cai JY, Schulte O, Xiong H (2007) The minimum consistent subset cover problem, its applications in data mining. In: KDD, pp 310–319
Geerts F, Goethals B, Mielikäinen T (2004) Tiling databases. In: Discovery science, pp 278–289
Gionis A, Mannila H, Seppänen JK (2004) Geometric, combinatorial tiles in 0-1 data. In: PKDD, pp 173–184
Han J, Kamber M (2006) Data mining: concepts, techniques, second edition. Morgan Kaufmann, San Francisco
Harper LH (1964) Optimal assignments of numbers to vertices. J Soc Ind Appl Math 12(1):131–135
Hartigan JA (1972) Direct clustering of a data matrix. J Am Stat Assoc 67(337):123–129
Jin R, Xiang Y, Fuhry D, Dragan FF (2008) Overlapping matrix pattern visualization: a hypergraph approach. In: ICDM, pp 313–322 (© IEEE, 2008. doi:10.1109/ICDM.2008.102)
Jin R, Xiang Y, Liu L (2009) Cartesian contour: a concise representation for a collection of frequent sets. In: KDD, pp 417–426
Johnson D, Krishnan S, Chhugani J, Kumar S, Venkatasubramanian S (2004) Compressing large boolean matrices using reordering techniques. In: VLDB, pp 13–23
Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer Verlag, New York
Lakshmanan LVS, Ng RT, Wang CX, Zhou X, Johnson TJ (2002) The generalized mdl approach for summarization. In: VLDB, pp 766–777
Li T (2005) A general model for clustering binary data. In: KDD, pp 188–197
Madeira SC, Oliveira AL (2004) Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Trans Comput Biol Bioinf 1(1):24–45
Minoux M (1977) Accelerated greedy algorithms for maximizing submodular set functions. In: the 8th IFIP Conference on Optimization Techniques
Mirkin B (1996) Mathematical classification and clustering. Kluwer Academic Publishers, Boston
Peeters R (2003) The maximum edge biclique problem is np-complete. Discret Appl Math 131(3):651–654
Pei J, Dong G, Zou W, Han J (2004) Mining condensed frequent-pattern bases. Knowl Inf Syst 6(5): 570–594
Richardson TJ, Urbanke RL (2008) Modern coding theory. Cambridge University Press, Cambridge
Safro I, Ron D, Brandt A (2006) Graph minimum linear arrangement by multilevel weighted edge contractions. J Algorithms 60(1):24–41
Siebes A, Vreeken J, van Leeuwen M (2006) Item sets that compress. In: SDM
Steinbach M, Tan P-N, Kumar V (2004) Support envelopes: a technique for exploring the structure of association patterns. In: KDD, pp 296–305
van Leeuwen M, Vreeken J, Siebes A (2006) Compression picks item sets that matter. In: PKDD, pp 585–592
Vreeken J, van Leeuwen M, Siebes A (2007) Characterising the difference. In: KDD, pp 765–774
Wang J, Karypis G (2006) On efficiently summarizing categorical databases. Knowl Inf Syst 9(1):19–37
Xiang Y, Jin R, Fuhry D, Dragan FF (2008) Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In: KDD, pp 758–766 (© ACM, 2008. http://doi.acm.org/10.1145/1401890.1401981)
Xin D, Han J, Yan X, Cheng H (2005) Mining compressed frequent-pattern sets. In: VLDB, pp 709–720