

ALGORITHMS FOR PATTERN MINING IN THE PRESENCE OF
TOUGH BLOCK CONSTRAINTS

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Krishna Gade

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

September 2004

©Krishna Gade 2004.

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of masters thesis by

Krishna Gade

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examination committee have been made.

Prof. George Karypis

09/02/04

GRADUATE SCHOOL

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. George Karypis for his continuous support and guidance ever since I arrived at the University of Minnesota. I thank him for helping me acquire key problem solving skills and instilling confidence in me to pursue future research. I am especially grateful for his patience and understanding in dealing with all the silly mistakes I have made. I thank him for not giving up on me, even when things looked not so good. I would like to thank Prof. Vipin Kumar for his help and advice at all the times. I would also like to thank Prof. Gedas Adomavicius for taking time to be on my defense committee.

This work was supported in part by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and ACI-0312828; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

TABLE OF CONTENTS

LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER	
1	1
1.1 Introduction	1
2	5
2.1 Definitions and Notation	5
2.2 Problem Definition	7
2.3 Related Research	10
3	12
3.1 Matrix-Projection based Pattern Mining	12
3.1.1 Frequent Pattern Enumeration	12
3.1.2 Frequent Closed Pattern Mining	14
4	17
4.1 Valid and Closed Block Mining with Tough Constraints	17
4.2 Pruning Methods	19
4.2.1 Column Pruning	19
Block Size	19
Block Sum	20
Block Similarity	21
4.2.2 Row Pruning	23
Block Size	24
Block Sum	24
Block Similarity	25
4.2.3 Matrix Pruning	26
Block Size	26
Block Sum	27
Block Similarity	28
5	29
5.1 Experiments	29
5.1.1 Test Environment and Datasets	29
5.1.2 Experimental Results	30
Closed vs. Non-Closed block pattern mining	30

Comparison with CLOSET+	30
Effectiveness of the Pruning Methods	31
Scalability Study	33
5.1.3 Application - Micro Concept Discovery	34
5.2 Conclusion and Future Work	37
BIBLIOGRAPHY	37

LIST OF TABLES

Table

1.1 TRANSACTION HISTORY OF ITEMS I1,I2,...,I5	3
5.1 DATASET CHARACTERISTICS.	30
5.2 SUMMARY OF DOCUMENT DATASETS USED FOR THE APPLICATION	33

LIST OF FIGURES

Figure

- 3.1 (A) A TRANSACTION DATABASE WITH $\theta \geq 0.5$; (B) THE PATTERN TREE. 13

CHAPTER 1

1.1 Introduction

Finding frequent patterns in large databases is a fundamental data mining task with extensive applications to many areas including association, correlation, and causality rule discovery, association-rule-based classification, and feature-based clustering. As a result, a vast amount of research has focused on this problem resulting in the development of numerous efficient algorithms. This research has been primarily focused on finding frequent patterns corresponding to itemsets and sequences, but the ubiquitous nature of the problem has also resulted in the development of various algorithms that find frequent spatial, geometric, and topological patterns, as well.

In recent years, researchers have recognized that in many application areas and problem settings frequency is not the best measure to use in determining the significance of a pattern as it depends on a number of other parameters such as the type of items that it contains, the length of the pattern, or various numerical attributes associated with the individual items. In such cases, even though frequent pattern discovery algorithms can still be used as a pre-processing step to identify a set of candidate patterns that are subsequently pruned by taking into account the additional parameters, they tend to lead to inefficient algorithms as a large number of the discovered patterns will eventually get eliminated. To address this problem, various *constrained* frequent pattern mining problem formulations have been developed that enable the user to focus on mining patterns with a rich class of constraints that capture the application semantics. This includes algorithms that assign different frequency constraints to the individual items [15, 32] and algorithms that find the itemsets X that satisfy various quantitative constraints of the form $\text{avg}(X) \leq v$, $\text{avg}(X) \geq v$, $\text{sum}(X) \leq v$, or $\text{sum}(X) \geq v$ [12]. The key property of these itemset constraints is that they are usually (or can be converted to) anti-monotone or monotone, making it possible to develop computationally efficient algorithms to find the corresponding patterns.

In this paper we introduce a new class of constraints referred to as *block constraints*, which determine the significance of an itemset pattern by considering the dense block that is formed by the pattern's items and its associated set of transactions. Specifically, we focus on three different block constraints called *block size*, *block sum*, and *block similarity*. The block size constraint applies to binary datasets, the block sum constraint applies to datasets in which each instance of an item has a non-negative value associated with it that can vary across transactions, and the block similarity constraint applies to datasets in which each transaction corresponds to a vector-space representation of an object and the similarity between these objects is measured by the cosine of their vectors. According to the block size constraint, a pattern is interesting if the size of its dense block (obtained by multiplying the length of the itemset and the number of its supporting transactions) is greater than a user-specified threshold. Analogously, according to the block sum constraint, a pattern is interesting if the sum of the values of its dense block is greater than a user-specified threshold. Finally, according to the block similarity constraint a pattern is interesting if its dense block accounts for a certain user-specified fraction of the overall similarity between the objects in the entire dataset.

Finding patterns satisfying the above constraints has applications in a number of different areas. For example, in the context of market-basket analysis, the block-size and block-sum constraints can be used to find the itemsets that account for a certain fraction of the overall quantities sold or revenue/profit generated, respectively, whereas in the context of document clustering, the block similarity constraint can be used to identify the set of terms that bring a set of documents together and thus correspond to thematically related words (commonly referred to as *micro-concepts* [13]).

Consider the transaction history data for a store. Table 1.1 shows the transaction records (T1,T2,...,T4) for the the 5 items (I1,I2,I3,I4,I5) , with each cell containing the price of the item in dollars.

Suppose, now a data analyst wanted to find the set of items which collectively contribute more than \$35 to the store sales. Traditional itemset constraints like

	I1	I2	I3	I4	I5	Other Items
T1	50	0	3	10	2	...
T2	0	5	3	10	2	...
T3	50	5	0	0	2	...
T4	0	5	3	10	2	...
...

Table 1.1: Transaction History of items I1,I2,...,I5

$sum(X) \geq 35$, where X is the itemset, find only the pattern $\{I1, I5\}$ supported by the transactions $\{T1, T3\}$. But if we observe there can be many other patterns such as $\{I4, I5\}$ supported by $\{T1, T2, T4\}$ that contribute \$36 and hence are valid. The reason why the itemset constraint fails to find them is because $\{I4, I5\}$ don't satisfy the condition (contribute only \$12) individually in each transaction, but together they contribute more than the requisite amount. One can argue, that the itemset discovery algorithms can be modified to find such remaining patterns by just accounting for their $price(i)$ whenever the item i is present and finding the total price of the supporting set. But this process will fail if the value associated with the item (which can be anything other than price) changes from transaction to transaction. Hence, there is a need for algorithms that can efficiently mine such constraints.

Developing computationally efficient algorithms to find these block constraints is particularly challenging because unlike the different itemset-based constraints studied earlier, these block constraints are *tough* [21] as they are neither anti-monotone, monotone, nor convertible [21]. To overcome this problem, we introduce a new class of pruning methods that can be used to significantly reduce the overall search space and make it possible to develop computationally efficient block constraint mining algorithms. Specifically, we focus on the problem of finding the *closed* itemsets satisfying the proposed block constraints and present a projection based mining framework, called CBMINER that takes advantage of a matrix-based representation of the dataset. CBMINER pushes deeply the various block constraints into closed pattern mining by using three novel classes of pruning methods called *column pruning*, *row pruning*, and *matrix pruning* that when combined lead to dramatic performance improvements. We present an extensive experimental evaluation using various datasets

that shows that CBMINER not only generates more concise result set, but also is much faster than the traditional frequent closed itemset mining algorithms. Moreover, we present an interesting application in the context of document clustering that illustrates the usefulness of the block similarity constraint in micro-concept discovery.

The rest of the paper is organized as follows. Section 2.1 introduces some basic definitions and notations. Section 2.2 formulates the problem and motivates each one of the three block constraints. Section 2.3 describes some related work. Section 3.1 derives the framework for mining closed blocks, while Section 4.2 discusses in detail how to efficiently mine closed patterns with tough block constraints. The thorough performance study is presented in Section 5.1. Finally, Section 5.2 provides some concluding remarks.

CHAPTER 2

2.1 Definitions and Notation

A transaction database is a set of transactions, where each *transaction* is a 2-tuple containing a transaction id and a set of items. Let \mathcal{I} be the complete set of distinct items and \mathcal{T} be the complete set of transactions. Any non-empty set of items is also called an *itemset* and any set of transactions is called a *transaction set*. The frequency of an itemset X (denoted as $\text{freq}(X)$) is the number of transactions that contain all the items in X , while the support of X is defined as $\sigma(X) = \text{freq}(X)/|\mathcal{T}|$. For a given minimum support threshold θ ($0 < \theta \leq 1$), X is said to be *frequent* if $\sigma(X) \geq \theta$. A frequent pattern X is called *closed* if there exists no proper super-pattern of X with the same support as X . An itemset constraint C is a predicate on the power set $2^{\mathcal{I}}$, i.e., $C : 2^{\mathcal{I}} \rightarrow \{TRUE, FALSE\}$. An itemset constraint C is *anti-monotone* if for any itemset X that satisfies C , all the subsets of X also satisfy C , and C is *monotone* if all the supersets of X satisfy C . For example, the constraint $\sigma(X) \geq \theta$ is anti-monotone, while $\sigma(X) \leq \theta$ is monotone. An itemset constraint is *tough* if it is neither anti-monotone nor monotone, and cannot be converted to either anti-monotone or monotone constraint.

A *block* is defined as a 2-tuple $B = (I, T)$, consisting of a transaction set T , and an itemset I , such that T is the supporting set of I . The *size* of a block B is defined as $\text{BSize}(B) = |I| \times |T|$. A *weighted block* is a block $B = (I, T)$ with a weight function w defined on the cross-product of the transaction set and itemset, i.e., $w : 2^{\mathcal{I}} \times 2^{\mathcal{T}} \rightarrow \mathcal{R}^+$, where \mathcal{R}^+ is the set of positive real numbers. The *sum* of a weighted block B is defined as $\text{BSum}(B) = \sum_{t \in T, i \in I} w(t, i)$. A (weighted) block $B = (I, T)$ is said to be a (weighted) *closed block* if and only if there exists no other (weighted) block $B' = (I', T')$ such that $I' \supset I$ and $T' = T$. Given a (weighted) block $B = (I, T)$, a (weighted) block $B' = (I', T')$ is a proper *superblock* of B if $I' \supset I$ and $T' \subseteq T$. In such a case B is called a (weighted) proper *subblock* of B' . We will use $B' \supset B$ to denote that B' is a proper superblock of B and $B \subset B'$ to denote that B

is a proper subblock of B' .

A *block constraint* C is a predicate on the power set $2^{\mathcal{I} \times \mathcal{T}}$, i.e., $C : 2^{\mathcal{I} \times \mathcal{T}} \rightarrow \{TRUE, FALSE\}$. A block B is called a *valid block* for constraint C if it satisfies constraint C (i.e., $C(B)$ is *TRUE*). A block constraint C is a *tough constraint* if there is no dependency between the satisfaction/violation of a constraint by a block and the satisfaction/violation of the constraint by any of its superblocks or subblocks.

A *transaction-item matrix* \mathcal{M} is a matrix where each row r represents a transaction and each column c represents an item in \mathcal{T} such that the value of the (r, c) entry of the matrix, denoted by $\mathcal{M}(r, c)$ is one iff transaction r supports c , otherwise $\mathcal{M}(r, c)$ is zero. Similarly a *weighted transaction-item matrix* \mathcal{M} is a transaction-item matrix where for each row r and for each column c , $\mathcal{M}(r, c)$ is equal to $w(r, c)$ (where w is a positive weight function defined on all transaction-item pairs in \mathcal{T}). A (weighted) block $B = (I, T)$ can be redefined as a (weighted) dense *submatrix* of the (weighted) transaction-item matrix \mathcal{M} formed with the rows of T and columns of I such that $\forall r \in T$ and $\forall c \in I$ we have $\mathcal{M}(r, c) = 1$ ($\mathcal{M}(r, c) > 0$).

Given a pre-defined ordering of the columns of \mathcal{M} and a set p of columns in \mathcal{M} , a *p -projected matrix* w.r.t. \mathcal{M} , $\mathcal{M}|_p$, is defined as the submatrix of \mathcal{M} containing only the rows that support itemset p and the columns that appear after p in matrix \mathcal{M} . For any transaction t in $\mathcal{M}|_p$, its *size* is defined as the number of non-zero elements in its corresponding row of $\mathcal{M}|_p$ and will be denoted by $|t|$. For any column x of $\mathcal{M}|_p$, the matrix obtained by keeping only the rows of $\mathcal{M}|_p$ that contain x is denoted as $\mathcal{M}|_p^x$. For each matrix $\mathcal{M}|_p$ and $\mathcal{M}|_p^x$ we will denote their set of corresponding transactions and items as $\mathcal{T}|_p$, $\mathcal{T}|_p^x$, $\mathcal{I}|_p$, and $\mathcal{I}|_p^x$, respectively.

Given a set of m -dimensional vectors $\mathcal{A} = \{\vec{d}_1, \vec{d}_2, \dots, \vec{d}_n\}$, the *composite vector* of \mathcal{A} is denoted by \vec{D} and is defined to be $\sum_{\vec{d} \in \mathcal{A}} \vec{d}$. Given a weighted block $B = (I, T)$, the *composite vector of the block* is denoted by \vec{B}_I and is the $|\mathcal{I}|$ -dimensional vector obtained as follows. For each item $i \in I$, the i th dimension of \vec{B}_I , denoted by $\vec{B}_I(i)$, is equal to $\sum_{\forall t \in T} w(t, i)$, otherwise if $i \notin I$, $\vec{B}_I(i) = 0$. Also, given a p -projected matrix $\mathcal{M}|_p$, the *composite vector of an item x within $\mathcal{M}|_p$* is denoted by \vec{B}^x and is

the $|\mathcal{I}|$ -dimensional vector obtained from the transactions included in $\mathcal{T}|_p^x$ such that for every $i \in \mathcal{I}|_p$, $\vec{B}^x(i) = \sum_{\forall t \in \mathcal{T}|_p^x} w(t, i)$, otherwise if $i \notin \mathcal{I}|_p$, $\vec{B}^x(i) = 0$.

Given a matrix \mathcal{M} , the *column-sum* of column i in \mathcal{M} is denoted by $\text{csum}_{\mathcal{M}}(i)$ and is defined to be equal to the sum of the values of the column i of \mathcal{M} , i.e., $\text{csum}_{\mathcal{M}}(i) = \sum_t \mathcal{M}(t, i)$. Similarly, the *row-sum* of row t in \mathcal{M} is denoted by $\text{rsum}_{\mathcal{M}}(t)$ and is defined to be equal to the sum of the values of the row t of \mathcal{M} , i.e., $\text{rsum}_{\mathcal{M}}(t) = \sum_i \mathcal{M}(t, i)$.

2.2 Problem Definition

In this paper we develop efficient algorithms for finding valid closed blocks that satisfy certain tough block constraints. Specifically, we focus on three types of block constraints that are motivated and described in this section.

Block Size Constraint In the context of market-basket analysis we are often interested in finding the set of itemsets each of which accounts for a certain fraction of the overall number of transactions that was performed during a certain period of time. Given an itemset I and its supporting set T , the extent to which I will satisfy this constraint will depend on whether or not $|I| \times |T|$ is no less than the specified fraction. Finding this type of itemsets is the motivation behind the first block-constraint that we study, which focuses on finding all blocks $B = (I, T)$ whose size is no less than a certain threshold. Specifically, given a binary transaction database \mathcal{T} , the ***block-size constraint*** is defined as

$$\text{BSize}(B) \geq \theta N, \quad (2.1)$$

where $0 < \theta \leq 1$ and N is the total number of non-zeros in the transaction-item matrix of \mathcal{T} , i.e., $N = \sum_{t \in \mathcal{T}} |t|$.

Note that depending on the size of the itemsets associated with each valid block, the minimum required size of the corresponding transaction set will be different. Small itemsets will require larger transaction sets, whereas large itemsets will lead to valid blocks with smaller transaction sets. As a result, even if an itemset I is not

part of a valid block, an extension of I , I' , may become valid (e.g., cases in which the support of I' does not significantly decrease compared to the support of I). Similarly, an itemset I which is not part of any valid block may contain subsets that are part of some valid blocks (e.g., cases in which the support of the subset is significantly greater than the support of I). Consequently, the block-size constraint is a *tough* constraint as it is neither anti-monotone nor monotone, and cannot be converted to either anti-monotone or monotone constraints.

Block Sum Constraint In cases in which there is a non-negative weight associated with each individual transaction-item pair (e.g., sales or profit achieved by selling an item to a customer), in addition to finding all itemsets that satisfy a certain block-size constraint we may also be interested in finding the itemsets whose corresponding weighted blocks have a block-sum that is greater than a certain threshold. For example, in the context of market-basket analysis, these itemsets can be used to identify the product groups that account for a certain fraction of the overall sales, profits, *etc.* Motivated by this, the second block-constraint that we study extends the notion of the block-size constraint to weighted blocks. Formally, given a transaction database \mathcal{T} , and a weight function w the ***block-sum constraint*** is defined as

$$\text{BSum}(B) \geq \theta W, \quad (2.2)$$

where $0 < \theta \leq 1$ and W is the sum of the weights of all the transaction-item pairs in the database, i.e., $W = \sum_{t \in \mathcal{T}, i \in \mathcal{I}} w(t, i)$. Note that since the block-sum constraint is a generalization of the block-size constraint it also represents a *tough* constraint.

Block Similarity Constraint The last block constraint that we will study is motivated by the problem of finding groups of thematically related words in large document datasets, each potentially describing a different *micro-concept* present in the collection. One way of finding such groups is to analyze the document-term matrix associated with the dataset and find sets of words that satisfy either a user specified minimum support constraint or a block-size constraint (as defined earlier). However,

the limitation of these approaches is that they do not account for the weights that are often associated with the various words as a result of the widely used tf-idf (term-frequency—inverse document-frequency) vector-space model [26]. In general, groups of words that have higher weights will more likely represent a thematically coherent concept than words that have very low weights, even if the latter groups have higher support. This often happens with words that are common in almost all the documents and will be assigned very low weight due to their high document frequency [3].

One way of addressing this problem is to first apply the tf-idf model on each document vector, scale the resulting document vectors to be of the same length (e.g., unit length), and then find the groups of related words by using the previously defined block-sum constraint. However, within the context of the vector-space model, a more natural way of measuring the importance of a group of words is to look at how much they contribute to the overall similarity between the documents in the collection. In other words, the micro-concept discovery problem can be formulated as that of finding all groups of words such that the removal of each group from their supporting documents will decrease the aggregate similarity between the documents by a certain fraction. In general, groups of words that are large, supported by many documents, and have high weights will tend to contribute a higher fraction to the aggregate similarity and hence form *better* micro-concepts.

Discovering groups of words that satisfy the above property led us to develop the block-similarity constraint that is defined as follows. Let $\mathcal{A} = \{d_1, d_2, \dots, d_n\}$ be a set of n documents modeled by their unit-length tf-idf representation of the set of documents, let m be the distinct number of terms in \mathcal{A} , let $B = (I, T)$ be a weighted block with I being a set of words and T being its supporting set of documents, let S be the sum of the pairwise similarities between the documents in \mathcal{A} , and let S' be the sum of the pairwise similarities between the documents in \mathcal{A} obtained after zeroing-out the entries corresponding to block B . The *similarity of the block B* is defined to be the *loss* in the aggregate pairwise similarity resulting from removing B ,

i.e., $\text{BSim}(B) = S - S'$, and the *block-similarity constraint* is defined as

$$\text{BSim}(B) \geq \theta S, \quad (2.3)$$

where $0 < \theta \leq 1$.

In this paper, we will measure the similarity between two documents d_i and d_j in \mathcal{A} by computing the dot-product of their corresponding vectors \vec{d}_i and \vec{d}_j (i.e., $\text{sim}(d_i, d_j) = \vec{d}_i \cdot \vec{d}_j$). Since the documents in \mathcal{A} have already been scaled to be of unit length, this similarity measure is nothing more than the cosine of their respective vectors, which is used widely in information retrieval. The advantage of the dot-product-based similarity measure is that it allows us to easily and efficiently compute both S and S' . Specifically, if \vec{D} is the composite vector of \mathcal{A} , it can be shown that $S = \vec{D} \cdot \vec{D}$. Similarly, if $B = (I, T)$ is a weighted block of \mathcal{A} , and \vec{B}_I is its corresponding composite vector it can be shown that $S' = (\vec{D} - \vec{B}_I) \cdot (\vec{D} - \vec{B}_I)$. As a result, the similarity of a block $B = (I, T)$ is given by

$$\text{BSim}(B) = S - S' = 2\vec{D} \cdot \vec{B}_I - \vec{B}_I \cdot \vec{B}_I. \quad (2.4)$$

To simplify the presentation of the three block constraints and the associated algorithms, in the rest of this paper we will consider the set of documents \mathcal{A} as forming a weighted transaction-item matrix \mathcal{M} whose rows and columns correspond to the documents and terms of \mathcal{A} , respectively. As a result, each matrix entry $\mathcal{M}(i, j)$ will be equal to $\vec{d}_i(j)$ (i.e., the value in the d_i 's vector along the j th dimension).

2.3 Related Research

Efficient algorithms for finding frequent itemsets in large databases have been one of the key success stories in data mining research [2, 19, 5, 24, 10, 17]. One of the early computationally efficient algorithms was Apriori [2], which finds frequent itemsets of length l based on the previously mined frequent itemsets of length $(l-1)$. More

recently, a set of database-projection-based methods [1, 10, 22] have been developed that significantly reduce the complexity of finding frequent long patterns. This study extends the projection-based method to mine valid sub-matrices with tough block constraints.

The frequent itemset mining algorithms usually generate a large number of frequent itemsets when the support is low. To solve this problem, two general classes of techniques were proposed. The first is mining closed/maximal patterns. Typical examples include Max-Miner [24], A-close [20], MAFIA [7], CHARM [17], CFP-tree [16], and CLOSET+ [29]. The *redundant pattern pruning* and *column fusing* methods adopted by CBMINER have been popularly used in different forms by several previous studies [24, 33, 23, 7, 17, 29, 16]. The second class focuses on mining constrained patterns by integrating various anti-monotone, monotone, or convertible constraints. The constrained association rule mining problem was first considered in [25] but only for item specific constraints. Since then a number of different constrained frequent pattern mining algorithms have been proposed [4, 18, 21, 12, 6, 11, 8, 14]. All these algorithms concentrate on constrained itemset mining with various anti-monotone, monotone, succinct or convertible constraints.

Very recently some work [31] has been done to push aggregate constraints in the context of iceberg-cube computing. This algorithm mines aggregate constraints in the GROUP BY partitions of an SQL query by using a divide-and-approximate strategy. The algorithm makes use of the strategy to derive a sequence of weaker anti-monotone constraints for a given non-anti-monotone constraint to prune the nodes in the search tree. Recently the LPMiner algorithm [27] was proposed to mine itemsets with length-decreasing support constraints. It uses a novel **SVE** property to prune the unpromising transactions of the projected databases based on the length of the transactions. Later the **SVE** property has been used to mine sequences and closed itemsets with length decreasing support constraints [28, 30]. We also explore the **SVE** property in the context of mining closed patterns with block constraints in Section 4.2.2 to prune the unpromising rows of a prefix-projected matrix.

CHAPTER 3

3.1 Matrix-Projection based Pattern Mining

In this section we describe the `FREQPTRNMINER`, `CLSDPTRNMINER` algorithms, which form the basis of `BMINER` and `CBMINER` algorithms. `FREQPTRNMINER` follows the widely used projection-based pattern mining paradigm [1, 10, 22], which can be used to efficiently mine the complete set of frequent patterns in a depth-first search order and as we will see later, it can be easily adapted to mine valid block patterns. A key characteristic of `FREQPTRNMINER`, `CLSDPTRNMINER` (as well as `BMINER` and `CBMINER`) is that they represent the transaction database \mathcal{T} using the transaction-item matrix \mathcal{M} and employs a number of efficient sparse matrix storage and access schemes, allowing it to achieve high computational efficiency. For the remainder of this section we describe the basic structure of `FREQPTRNMINER` for the problem of enumerating all patterns satisfying a constant minimum support constraint and then introduce several pruning methods to accelerate the frequent closed pattern mining done by `CLSDPTRNMINER`. The extension of this algorithm for finding the closed blocks that satisfy the three tough block constraints described in Section 2.2 will be described later in Section 4.2.

3.1.1 Frequent Pattern Enumeration

Given a database, the complete set of itemsets can be organized into a lattice if the items are in a predefined order, and the problem of frequent pattern mining then becomes how to traverse the lattice to find the frequent ones. The `FREQPTRNMINER` algorithm adopts the depth-first search traversal and uses the downward closure property to prune the infrequent columns from further mining. Figure 3.1(a) shows a database example with a minimum support 0.5. If we remove the set of infrequent columns, $\{b,f,h,i,k,m\}$, and sort the set of frequent columns in frequency-increasing order, then part of the lattice (i.e., pattern tree) formed from column set $\{g,a,c,e,d\}$ can be organized into the one shown in Figure 3.1(b). Each node in the lattice is

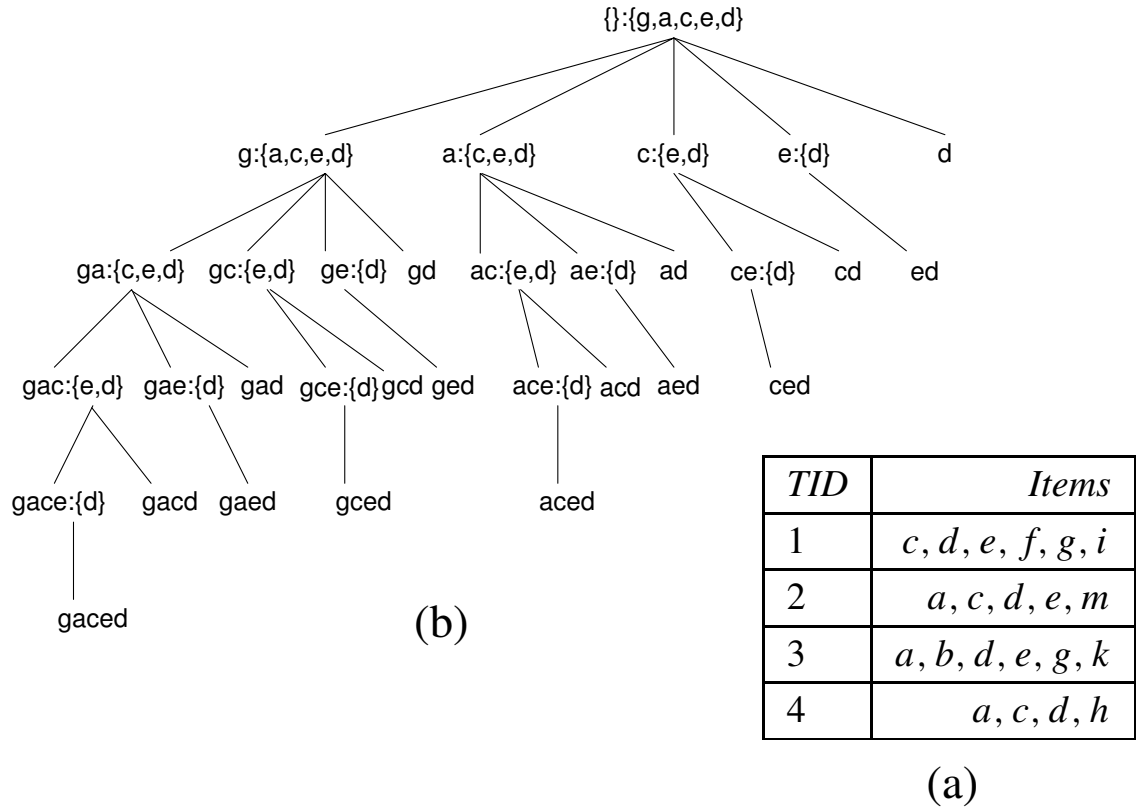


Figure 3.1: (a) A transaction database with $\theta \geq 0.5$; (b) The pattern tree.

labeled in the form $p:q$, where p is a prefix itemset and q is the set of local columns appeared in the p -projected matrix, $\mathcal{M}|_p$. At a certain node during the depth-first traversal of the lattice, if the corresponding prefix p is infrequent, we stop mining the sub-tree under this node. Otherwise, we report p as a frequent pattern, build its projected matrix, $\mathcal{M}|_p$, find its locally frequent columns in $\mathcal{M}|_p$ and use them to grow p to get longer itemsets.

To store the various projected matrices efficiently, we adopt the CSR *sparse* storage scheme [9]. The CSR format utilizes two one-dimensional arrays: the first stores the actual non-zero elements of the matrix in a row (or column) major order, and the second stores the indices corresponding to the beginning of each row (or column). To ensure that both the matrix projection as well as the column frequency counting are performed efficiently, we maintain both the row- and the column-based

representation of the matrix. The overall complexity of the algorithm depends on the two key steps of sorting and projecting. We used the radix sort algorithm to sort the column frequencies which has a time complexity that is linear in the number of columns being sorted, and because of our matrix-storage scheme, projecting the matrix on the column is linear on the number of non-zeros in the projected matrix. Our matrix-projection based pattern enumeration method shares some of the ideas with the recently developed array-projection based method [22], which was shown to achieve good performance, especially for sparse datasets.

Algorithm 3.1.1: $\text{FREQPTRNMINER}(p, \mathcal{M}|_p, \theta)$

```

Sort the columns of  $\mathcal{M}|_p$  in frequency increasing order
Prune the columns in  $\mathcal{M}|_p$  whose support is less than  $\theta$ 
if no column is frequent
  then return
for each column  $a \in \mathcal{M}|_p$ 
  do  $\left\{ \begin{array}{l} \text{Output the frequent pattern } p \cup \{a\} \\ \text{Project } \mathcal{M}|_p \text{ on } a \text{ to get } \mathcal{M}|_{p \cup \{a\}} \\ \text{FREQPTRNMINER}(p \cup \{a\}, \mathcal{M}|_{p \cup \{a\}}, \theta, H) \end{array} \right.$ 
return

```

3.1.2 Frequent Closed Pattern Mining

The above frequent pattern enumeration method can find the complete set of frequent itemsets. To get the set of frequent closed itemsets, we need to check whether a newly found itemset is closed or not and sift out the redundant (i.e., non-closed) ones. The pattern closure checking in CLSDPTRNMINER works as follows. We maintain the set of frequent closed itemsets mined so far in a hash-table H using the sum of the transaction-IDs of the supporting transactions as the hash-key [33, 17]. Upon getting a new itemset p , we check against the set of already mined closed itemsets which have the same hash-key value as the one derived from p 's sum of transaction-IDs, to see if there is any itemset that is a proper superset of p with the same support. If that is the case, p is non-closed, otherwise the union of p and the set of its local columns with the same support as p forms a closed itemset.

In the pattern enumeration process, some prefix itemsets or columns are unpromising to generate closed itemsets and thus can be pruned. CLSDPTRNMINER adopts two pruning methods, *redundant pattern pruning* and *column fusing* [24, 33, 23, 7, 17, 29]. Also, to make the matrix a more compact representation, we propose the *row fusing* method to compress the projected matrix.

1. **Redundant Pattern Pruning (RPP)** Once we find that a prefix itemset is non-closed, that is, it is a proper subset of another already mined closed itemset with the same support, it can be safely pruned, and the sub-tree under the node corresponding to this prefix will not be traversed.
2. **Column Fusing (CF)** This optimization performs two different tasks. First, it fuses the completely dense columns of the projected matrix $\mathcal{M}|_p$ to the prefix itemset p and removes them from $\mathcal{M}|_p$, and thus avoiding projections on them. Second, it fuses columns in $\mathcal{M}|_p$ that have identical supporting transaction sets into a single column, and removes the original columns from $\mathcal{M}|_p$. By fusing them, the algorithm reduces the number of projections that need to be performed, as it essentially allows for the pattern to grow by adding multiple columns in a single step.

By integrating the above optimization methods with the frequent pattern enumeration process, we get the CLSDPTRNMINER algorithm shown in Algorithm 3.1.2. It takes as input the current pattern p , the p -projected matrix $\mathcal{M}|_p$, the given minimum support θ , and the current hash-table H . The algorithm initially sorts the columns of $\mathcal{M}|_p$ and eliminates any infrequent columns and then proceeds to perform *Column Fusing*. After that it enters its main computational loop which extends p by adding each column $a \in \mathcal{M}|_p$, checks to see if $p \cup \{a\}$ can be pruned by comparing it against H (*Redundant Pattern Pruning*), projects $\mathcal{M}|_p$ on a , checks to see if $p \cup \{a\}$ is closed, and finally calls itself recursively for pattern $p \cup \{a\}$.

Algorithm 3.1.2: CLSDPTRNMINER($p, \mathcal{M}|_p, \theta, H$)

```

Sort the columns of  $\mathcal{M}|_p$  in frequency increasing order
Prune the columns in  $\mathcal{M}|_p$  whose support is less than  $\theta$ 
if no column is frequent
  then return
Do Column Fusing for the columns in  $\mathcal{M}|_p$ 
for each column  $a \in \mathcal{M}|_p$ 
  {
  if  $p \cup \{a\}$  is a Redundant Pattern
    then continue
  Project  $\mathcal{M}|_p$  on  $a$  to get  $\mathcal{M}|_{p \cup \{a\}}$ 
  do {
  if there is no dense column in  $\mathcal{M}|_{p \cup \{a\}}$ 
    then {
    Output the closed pattern  $p \cup \{a\}$ 
    Insert  $p \cup \{a\}$  into the hash-table  $H$ 
    CLSDPTRNMINER( $p \cup \{a\}, \mathcal{M}|_{p \cup \{a\}}, \theta, H$ )
  }
  }
return

```

CHAPTER 4

4.1 Valid and Closed Block Mining with Tough Constraints

Like the traditional frequent and closed pattern mining algorithms, `FREQPTRNMINER` and `CLSDPTRNMINER` work under the constant support threshold framework and use the downward closure property to prune infrequent columns. However, with tough block constraints, the nice properties derived from the anti-monotone (or monotone) constraints no longer hold to be used to prune search space. Designing effective pruning methods for tough block constraints is especially challenging. To address this challenge we developed three classes of pruning methods, called *column-pruning*, *row-pruning* and *matrix pruning*, which eliminate the unpromising columns, rows and projected matrices from mining. The specific details of these pruning methods are different for each of the three block constraints and will be described later in this section.

By incorporating these three pruning methods with the overall structure of `FREQPTRNMINER` and `CLSDPTRNMINER`, we can easily derive the `BMINER`, `CBMINER` algorithms that mine efficiently the set of all valid block and closed block patterns respectively. The pseudo code for `BMINER` is shown in Algorithm 4.1.1 and that of `CBMINER` is shown in Algorithm 4.1.2. Both of them take as input the current pattern p , its corresponding p -projected matrix $\mathcal{M}|_p$, and the block-constraint C that corresponds to either the block-size, block-sum, or block-similarity constraint. But the `CBMINER` algorithm takes the hash-table H that stores the valid closed blocks, as an extra input argument. Since they are derived from `FREQPTRNMINER`, `CLSDPTRNMINER` algorithms, they have many steps in common and for this reason we will only describe the key differences.

Algorithm 4.1.1: $\text{BMINER}(p, \mathcal{M}|_p, H, C)$

Sort the columns of $\mathcal{M}|_p$ in frequency increasing order

if matrix $\mathcal{M}|_p$ can be pruned

then return

Prune the columns in $\mathcal{M}|_p$

if no column is valid

then return

for each column $a \in \mathcal{M}|_p$

 do {

 let $B = (p \cup \{a\}, \mathcal{T}|_p^a)$

 if $C(B) = \text{TRUE}$

then { **Output** the valid block B

Project $\mathcal{M}|_p$ on a to get $\mathcal{M}|_{p \cup \{a\}}$

Prune the rows of $\mathcal{M}|_{p \cup \{a\}}$

$\text{BMINER}(p \cup \{a\}, \mathcal{M}|_{p \cup \{a\}}, H, C)$

 }

return

Algorithm 4.1.2: $\text{CBMINER}(p, \mathcal{M}|_p, H, C)$

Sort the columns of $\mathcal{M}|_p$ in frequency increasing order

if matrix $\mathcal{M}|_p$ can be pruned

then return

Prune the columns in $\mathcal{M}|_p$

if no column is valid

then return

Do **Column Fusing** for the columns in $\mathcal{M}|_p$

for each column $a \in \mathcal{M}|_p$

 do {

 if $p \cup \{a\}$ is a **Redundant Pattern**

then continue

 let $B = (p \cup \{a\}, \mathcal{T}|_p^a)$

Project $\mathcal{M}|_p$ on a to get $\mathcal{M}|_{p \cup \{a\}}$

 if \nexists dense column in $\mathcal{M}|_{p \cup \{a\}}$ and $C(B) = \text{TRUE}$

then { **Output** the closed block B

Insert B into the hash-table H

Prune the rows of $\mathcal{M}|_{p \cup \{a\}}$

$\text{CBMINER}(p \cup \{a\}, \mathcal{M}|_{p \cup \{a\}}, H, C)$

 }

return

The first difference has to do with the pruning methods. Specifically, instead of using the constant support-based column pruning, BMINER , CBMINER use the newly proposed *column-pruning*, *row-pruning* and *matrix pruning* methods, which are derived from the tough block constraints. The second difference has to do with

the implementation of the column fusion and row fusion optimizations for the block-sum and block-similarity constraints. In the case of the block-sum constraint, the values of the fused columns (and rows) correspond to the sum of the values of their constituent columns (and rows). This ensures that the resulting *fused* matrix contains all necessary information to correctly evaluate the constraints. In the case of the block-similarity constraint, since the correct evaluation of the constraints requires access to the individual column- and row-values, we do not perform any column/row fusion.

Following we will introduce in detail the three pruning methods, *column-pruning*, *row-pruning* and *matrix pruning*, in terms of the three different block constraints.

4.2 Pruning Methods

4.2.1 Column Pruning

Given a prefix itemset p and its projected matrix $\mathcal{M}|_p$, the idea behind column pruning is to identify for each column $x \in \mathcal{M}|_p$ a *necessary* condition that must be satisfied such that there is a *valid* block $B = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ for which γ is a subset of the columns in $\mathcal{M}|_p$ and $x \in \gamma$. Using this condition, we can then eliminate from $\mathcal{M}|_p$ all the columns that do not satisfy it, as these columns cannot be part of a valid block that contain p . Note that for each column x that we eliminate, we prevent the exploration of the sub-tree associated with the pattern $p \cup \{x\}$, thus, significantly reducing the overall search space.

Block Size

The necessary condition for the block-size constraint is encapsulated in the following lemma (Refer to Section 2.1 for a description of the notation used).

Lemma 1 (*Block-Size Column Pruning*) *Let p be a pattern and x a column in $\mathcal{M}|_p$. Then in order for x to be part of a valid block that satisfies the block-size constraint of Equation 2.1 and is obtained from extending p by adding columns from*

$\mathcal{M}|_p$, the following must hold:

$$\text{BSize}(p, \mathcal{T}|_p^x) + \sum_{t \in \mathcal{T}|_p^x} |t| \geq \theta N. \quad (4.1)$$

Proof: Let γ be any arbitrary set of columns in $\mathcal{M}|_p$ such that $x \in \gamma$ and the block $B = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ satisfies the block-size constraint. Then from the definition of the block-size constraint we have that

$$|p \cup \gamma| \times |\mathcal{T}|_{p \cup \gamma} \geq \theta \times N.$$

Also, because $\mathcal{T}|_p^x \supseteq \mathcal{T}|_{p \cup \gamma}$ and for $\forall t \in \mathcal{T}|_{p \cup \gamma}, |t| \geq |\gamma|$, the following holds:

$$\sum_{t \in \mathcal{T}|_p^x} (|p| + |t|) \geq \sum_{t \in \mathcal{T}|_{p \cup \gamma}} (|p| + |t|) \geq |p \cup \gamma| \times |\mathcal{T}|_{p \cup \gamma}.$$

Equation 4.1 can be obtained by combining the above two inequalities and using the fact that $\text{BSize}(p, \mathcal{T}|_p^x) = |p| \times |\mathcal{T}|_p^x$. □ □

For each column in $\mathcal{M}|_p$, Equation 4.1 can be evaluated by adding up the lengths of the rows that it supports. These sums can be computed for all the columns by performing a single scan of the p -projected matrix.

Block Sum

The necessary condition for the block-sum constraint is similar in nature to that of the block-size constraint and is encapsulated in the following lemma.

Lemma 2 (*Block-Sum Column Pruning*) *Let p be a pattern and x a column of $\mathcal{M}|_p$. Then in order for x to be part of a valid block that satisfies the block-sum constraint of Equation 2.2 and is obtained by extending p with columns in $\mathcal{M}|_p$, the*

following must hold:

$$\text{BSum}(p, \mathcal{T}|_p^x) + \sum_{t \in \mathcal{T}|_p^x, j \in \mathcal{I}|_p} \mathcal{M}|_p(t, j) \geq \theta W. \quad (4.2)$$

This lemma can be proved in a similar way to Lemma 1 and the actual proof is omitted. Note that the summation on the left-hand-side of Equation 4.2 is nothing more than the sum of the non-zero elements of each row in $\mathcal{T}|_p^x$.

The various quantities required to evaluate Equation 4.2 can be computed efficiently by performing a single scan of the block $(p, \mathcal{T}|_p^x)$ to compute the sum of each row, and two scans of the matrix $\mathcal{M}|_p$. The first scan will compute the sum of the non-zero elements of each row, and the second scan will compute the summation term in Equation 4.2 for each column.

Block Similarity

Let \vec{D} be the composite vector of \mathcal{T} and consider a p -projected weighted matrix $\mathcal{M}|_p$. The necessary condition for the block-similarity constraint is encapsulated in the following lemma.

Lemma 3 (*Block-Similarity Column Pruning*) *Let p be a pattern, $(p, \mathcal{T}|_p)$ its corresponding block, and x a column of $\mathcal{M}|_p$. Then in order for x to be part of a block that satisfies the block-similarity constraint of Equation 3 and is obtained by extending p with columns in $\mathcal{M}|_p$, the following must hold:*

$$2\vec{D} \cdot (\vec{B}^x + \vec{B}_p) \geq \theta S \quad (4.3)$$

Proof: Let γ be any arbitrary set of columns in $\mathcal{M}|_p$ such that $x \in \gamma$ and the block $B = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ satisfies the block-similarity constraint, and let $\vec{B}_{p \cup \gamma}$ be its corresponding composite vector. Then from the definition of the block-similarity

constraint we have that

$$2\vec{D} \cdot \vec{B}_{p \cup \gamma} - \vec{B}_{p \cup \gamma} \cdot \vec{B}_{p \cup \gamma} \geq \theta S$$

Also, because $\mathcal{T}|_p \supseteq \mathcal{T}|_p^x \supseteq \mathcal{T}|_{p \cup \gamma}$, the following holds:

$$2\vec{D} \cdot (\vec{B}_p + \vec{B}^x) \geq 2\vec{D} \cdot \vec{B}_{p \cup \gamma} \geq 2\vec{D} \cdot \vec{B}_{p \cup \gamma} - \vec{B}_{p \cup \gamma} \cdot \vec{B}_{p \cup \gamma}$$

Hence the above two in-equalities prove the lemma. \square

For each column of $\mathcal{M}|_p$, evaluating the above equation incurs a computational cost equivalent to one scan of the p -projected matrix, which is very costly. So, we make use of the following lemma, which approximates Equation 7.

Lemma 4 (*Approximate Block-Similarity Column Pruning*) *Let ξ be the maximum value across the m dimensions of vector \vec{D} and τ be the maximum row-sum over all the rows of the p -projected matrix $\mathcal{M}|_p$. Then in order for x to be part of a block that satisfies the block-similarity constraint of Equation 3 and is obtained by extending p with columns in $\mathcal{M}|_p$, the following must hold:*

$$\text{freq}(x) \geq \frac{\theta S - 2\vec{D} \cdot \vec{B}_p}{2\xi\tau} \quad (4.4)$$

Proof: If ζ be the maximum row-sum over all the rows of $\mathcal{M}|_p^x$ we can then rewrite the dot product $2\vec{D} \cdot \vec{B}^x$ as follows:

$$\begin{aligned} 2\vec{D} \cdot \vec{B}^x &= 2 \sum_{i \in \mathcal{I}|_p^x} (\vec{D}(i) \times \text{csum}_{\mathcal{M}|_p^x}(i)) \\ &\leq 2\xi \sum_{i \in \mathcal{I}|_p^x} (\text{csum}_{\mathcal{M}|_p^x}(i)) \\ &= 2\xi \sum_{t \in \mathcal{T}|_p^x} (\text{rsum}_{\mathcal{M}|_p^x}(t)) \\ &\leq 2\xi\zeta \text{freq}(x) \leq 2\xi\tau \text{freq}(x). \end{aligned}$$

Combining this inequality with Equation 7 proves the lemma. \square

In a single scan of the projected matrix, we can compute the frequency of all its columns along with the value of τ . Hence the complexity is of the order of the size of the projected matrix.

4.2.2 Row Pruning

Given a pattern p and its projected matrix $\mathcal{M}|_p$, the idea behind row pruning is to identify for each row $t \in \mathcal{M}|_p$ a *necessary* condition that must be satisfied such that there is a *valid* block $B = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ for which $\gamma \subseteq t$. Using this condition, we can then eliminate from $\mathcal{M}|_p$ all the rows that do not satisfy it, as these rows cannot be part of a valid block that contain p . By eliminating such rows we reduce the size of $\mathcal{M}|_p$ and thus reduce the amount of time required to perform subsequent projections and enhance future column pruning operations.

To derive such conditions we make use of the *Smallest Valid Extension* (SVE) principle, originally introduced in [27] for finding itemsets with length-decreasing support constraint. In the context of block constraints considered in this paper, the smallest valid extension of a prefix p is defined as the length of the smallest possible extension γ to p (where γ is a set of columns in $\mathcal{M}|_p$), such that the resulting block $B = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ is valid for a given constraint C . That is,

$$\text{SVE}(p) = \min_{\gamma \subseteq \mathcal{I}|_p} \{|\gamma| \mid C(p \cup \gamma, \mathcal{T}|_{p \cup \gamma}) = \text{TRUE}\}.$$

Knowing the SVE of a pattern, we can then eliminate all rows whose length is smaller than the SVE value. Note that the SVE of a pattern that already corresponds to a valid block will be by definition zero. For this reason, the row-pruning is only applied when the pattern p does not correspond to a valid block.

In the rest of this section we describe how to obtain such SVE-based necessary conditions for the block-size, block-sum, and block-similarity constraints.

Block Size

The SVE of a pattern p for the block-size constraint is given by the following lemma.

Lemma 5 (*Block-Size Row Pruning*) *Let p be a pattern such that $B = (p, \mathcal{T}|_p)$ does not satisfy the block-size constraint. Then the smallest valid extension of p for the block-size constraint of Equation 2.1 is*

$$\text{SVE}(p) \geq \frac{\theta N - \text{BSize}(B)}{|\mathcal{T}|_p}. \quad (4.5)$$

Proof: We will show that it holds by contradiction. Assume that there is a set of items γ such that the block $B' = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ is valid and

$$|\gamma| < \frac{\theta N - \text{BSize}(B)}{|\mathcal{T}|_p}. \quad (4.6)$$

Because $|\mathcal{T}|_{p \cup \gamma} \leq |\mathcal{T}|_p$, we have that

$$\text{BSize}(B') = (|p \cup \gamma|) \times |\mathcal{T}|_{p \cup \gamma} \leq \text{BSize}(B) + |\gamma| \times |\mathcal{T}|_p. \quad (4.7)$$

Combining Equation 4.6 and Equation 4.7 we have

$$\text{BSize}(B') < \theta N,$$

indicating that B' does not satisfy the block-size constraint, violating our initial assumption about B' . This means that p needs to be extended by adding at least $(\theta N - \text{BSize}(B))/|\mathcal{T}|_p$ items. □

The complexity of computing the $\text{SVE}(p)$ is $\Theta(1)$.

Block Sum

The SVE of a pattern p for the block-sum constraint is given by the following lemma.

Lemma 6 (Block-Sum Row Pruning) *Let p be a pattern such that $B = (p, \mathcal{T}|_p)$ does not satisfy the block-sum constraint, and z be the maximum column-sum over all columns of $\mathcal{M}|_p$. Then the smallest valid extension of p for the block-sum constraint of Equation 2.2 is*

$$\text{SVE}(p) \geq \frac{\theta W - \text{BSum}(B)}{z}. \quad (4.8)$$

Proof: As with Lemma 5 we will show that it holds by contradiction. Assume that there is a set of items γ such that the block $B' = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ is valid and

$$|\gamma| < \frac{\theta W - \text{BSum}(B)}{z}. \quad (4.9)$$

Because $|\mathcal{T}|_{p \cup \gamma}| \leq |\mathcal{T}|_p|$, each column of γ must contribute no greater than z to the block-sum of B' , we have that

$$\text{BSum}(B') \leq \text{BSum}(B) + |\gamma|z. \quad (4.10)$$

Combining Equation 4.9 and Equation 4.10 we have that

$$\text{BSum}(B') < \theta W,$$

indicating that B' does not satisfy the block-sum constraint, violating our initial assumption about B' . This means that p needs to be extended by adding at least $(\theta W - \text{BSum}(B))/z$ items. □

The complexity of computing the $\text{SVE}(p)$ is of the order of the size of the projected matrix as we need a scan of the projected matrix to compute the maximum of the column-sums.

Block Similarity

Let \vec{D} be the composite vector of \mathcal{T} and consider a p -projected weighted matrix $\mathcal{M}|_p$. The *column-similarity* of column x in $\mathcal{M}|_p$ is denoted by $\text{csim}_{\mathcal{M}|_p}(x)$ and is defined

to be equal to $2\vec{D}(x)\text{csum}_{\mathcal{M}|_p}(x) - \text{csum}_{\mathcal{M}|_p}^2(x)$. Given this definition, the SVE of a pattern p for the block-similarity constraint is given by the following lemma.

Lemma 7 (*Block-Similarity Row Pruning*) *Let p be a pattern such that $B = (p, \mathcal{T}|_p)$ does not satisfy the block-similarity constraint, and z is the maximum column-similarity over all columns of $\mathcal{M}|_p$. Then the smallest valid extension of p for the block-similarity constraint of Equation 2.3 is*

$$\text{SVE}(p) \geq \frac{\theta S - \text{BSim}(B)}{z}. \quad (4.11)$$

The proof is similar to that used for Lemma 6. In particular, each column of $\mathcal{M}|_p$ can contribute at most z to the overall block-similarity, and thus p needs to grow by adding at least $(\theta S - \text{BSim}(B))/z$ items. The actual details of the proof are omitted. The complexity of computing the $\text{SVE}(p)$ is identical to that for the block-sum constraint.

4.2.3 Matrix Pruning

Given a prefix itemset p and its projected matrix $\mathcal{M}|_p$, the column pruning and row pruning methods are very effective in pruning some unpromising columns and rows from $\mathcal{M}|_p$. However, in many cases the whole projected matrix $\mathcal{M}|_p$ cannot be used to generate any valid block patterns and thus can be pruned. Hence we developed another class of pruning method called *matrix pruning* in order to further prune the search space in terms of the *block size*, *block sum*, and *block similarity* constraints.

Block Size

The necessary condition for the block-size constraint is encapsulated in the following lemma.

Lemma 8 (*Block-Size Matrix Pruning*) *Let p be a pattern and t a transaction in $\mathcal{M}|_p$. Then in order for $\mathcal{M}|_p$ to be used to generate any valid block that satisfies*

the block-size constraint of Equation 2.1 and is obtained by extending p with some columns in $\mathcal{M}|_p$, the following must hold:

$$\text{BSize}(p, \mathcal{T}|_p) + \sum_{t \in \mathcal{T}|_p} |t| \geq \theta N. \quad (4.12)$$

Proof: Let γ be any arbitrary set of columns in $\mathcal{M}|_p$ such that the block $B = (p \cup \gamma, \mathcal{T}|_{p \cup \gamma})$ satisfies the block-size constraint, that is:

$$|p \cup \gamma| \times |\mathcal{T}|_{p \cup \gamma} \geq \theta \times N.$$

Because $\mathcal{T}|_{p \cup \gamma} \subseteq \mathcal{T}|_p$ and for $\forall t \in \mathcal{T}|_{p \cup \gamma}, |t| \geq |\gamma|$, the following holds:

$$\sum_{t \in \mathcal{T}|_p} (|p| + |t|) \geq \sum_{t \in \mathcal{T}|_{p \cup \gamma}} (|p| + |t|) \geq |p \cup \gamma| \times |\mathcal{T}|_{p \cup \gamma}.$$

Equation 4.12 can be obtained by combining the above two inequalities and using the fact that $\text{BSize}(p, \mathcal{T}|_p) = |p| \times |\mathcal{T}|_p$. □ □

The sums in Equation 4.12 can be computed by a single scan of the p -projected matrix $\mathcal{M}|_p$.

Block Sum

The necessary condition for the block-sum constraint is stated in the following lemma.

Lemma 9 (*Block-Sum Matrix Pruning*) *Let p be a pattern, x a column in $\mathcal{M}|_p$, and t a transaction in $\mathcal{M}|_p$. Then in order for $\mathcal{M}|_p$ to be used to generate any valid block that satisfies the block-sum constraint of Equation 2.2 and is obtained by extending p with some columns in $\mathcal{M}|_p$, the following must hold:*

$$\text{BSum}(p, \mathcal{T}|_p) + \sum_{t \in \mathcal{T}|_p, x \in \mathcal{I}|_p} \mathcal{M}|_p(t, x) \geq \theta W. \quad (4.13)$$

This lemma can be proved in a similar way to Lemma 8 and the actual proof is omitted. Note that the summation on the left-hand-side of Equation 4.13 is nothing more than the sum of the non-zero elements of each row in $\mathcal{T}|_p$ and can be computed in one scan of the p -projected matrix.

Block Similarity

Using the definition of the *column-similarity* introduced in Section 4.2.2, the necessary condition for the block-similarity constraint can be stated as follows:

Lemma 10 (*Block-Similarity Matrix Pruning*) *Let p be a pattern, x a column of $\mathcal{M}|_p$, and $\text{csim}_{\mathcal{M}|_p}(x)$ the column-similarity of x in $\mathcal{M}|_p$. Then in order for $\mathcal{M}|_p$ to be used to generate any valid blocks that satisfy the block-similarity constraint of Equation 3 and is obtained from extending p with some columns in $\mathcal{M}|_p$, the following must hold:*

$$\text{BSim}(p, \mathcal{T}|_p) + \sum_{x \in \mathcal{I}|_p} \text{csim}_{\mathcal{M}|_p}(x) \geq \theta S \quad (4.14)$$

The proof is similar to that used for Lemma 6.8. In particular, each column x of $\mathcal{M}|_p$ can contribute at most $\text{csim}_{\mathcal{M}|_p}(x)$ to the overall block-similarity and thus the whole matrix $\mathcal{M}|_p$ contributes at most the sum of the column-similarities of its columns. The actual details of the proof are omitted. The column-similarities of all the columns can be computed in a single scan of the p -projected matrix.

CHAPTER 5

5.1 Experiments

5.1.1 Test Environment and Datasets

In this section, we evaluate CBMINER for all three constraints. All the experiments were performed on a 2GHz Intel P4 processor with 2GB of memory running Linux. CBMINER was implemented in C and all times reported are in seconds. Since there are no existing closed block-constraint mining algorithms, we chose one of the most recently developed frequent closed itemset mining algorithms, CLOSET+ [29], for our comparisons. We compared CBMINER with a Linux executable version of CLOSET+, by providing the minimum frequency of the valid closed block patterns generated by CBMINER as the absolute minimum support threshold to CLOSET+. This ensures that CLOSET+ will discover all the patterns found by CBMINER. However, CLOSET+ will find additional patterns that do not satisfy the various block constraints. We compared the running times as well as the number of patterns generated by CBMINER with those of CLOSET+. We also evaluated the pruning methods that we proposed and their various possible combinations. We used five real datasets and several synthetic datasets to evaluate the algorithm performance, the effectiveness of the pruning methods, and the scalability in terms of the database size. The characteristics (number of transactions, number of items and the average(maximum) transaction lengths) of the datasets are shown in the Table 5.1.

Real datasets: The *gazelle* and *BMSWebView2* datasets contain the click-stream data from Gazelle.com. The *pumsb** dataset contains census data and *big-market* dataset contains the transaction information of a retail store. The *sports* dataset is a document dataset obtained from San Jose Mercury (TREC).

Synthetic datasets: The synthetic datasets were generated from IBM dataset generator, with average transaction lengths of 40,20,30 and frequent itemset lengths of 10,10,15 respectively. We used T40I10D100K for our comparisons with CLOSET+ as it was used in many previous performance studies and the remaining two datasets for

our scalability experiments. To test the scalability against the base size, we generated the dataset series T20I10Dx and T30I15Dx by varying the number of transactions from 200K to 1000K.

Data	# Trans	# Items	A.(M.)tran.len.
gazelle	59601	498	2.5(267)
BMSWebView2	77512	3340	4.6(161)
pumsb*	49046	2089	50.5(63)
big-market	838466	38336	3.12(90)
Sports	8580	126373	258.3(2344)
T40I10D100K	100000	1000	39.6(77)

Table 5.1: Dataset Characteristics.

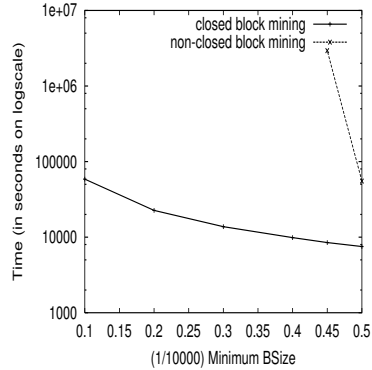
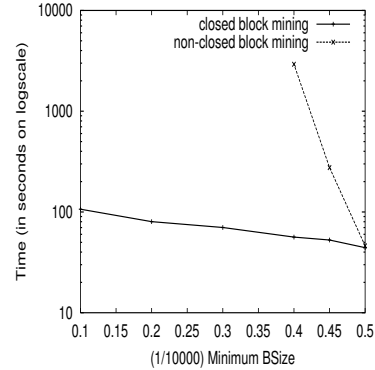
5.1.2 Experimental Results

Closed vs. Non-Closed block pattern mining

In this section we compare the performance and the number of patterns generated by CBMINER and BMINER algorithms shown in Algorithm 4.1.2 and Algorithm 4.1.1 respectively. The CBMINER algorithm shown in Algorithm 4.1.2 only finds closed valid block patterns by pruning the redundant patterns. However, BMINER algorithm finds all the valid block patterns. Our experiments show that in most cases CBMINER mines more concise result set and runs much faster than BMINER for all the three block constraints we studied, thus mining closed block patterns is a more desirable paradigm than mining all block patterns. Here we only report the comparison result for one dataset, *big-market*. As Fig. 2 and Fig.3 show that for dataset *big-market* and BSize constraint CBMINER finds orders of magnitude fewer patterns than BMINER while it can be orders of magnitude faster.

Comparison with CLOSET+

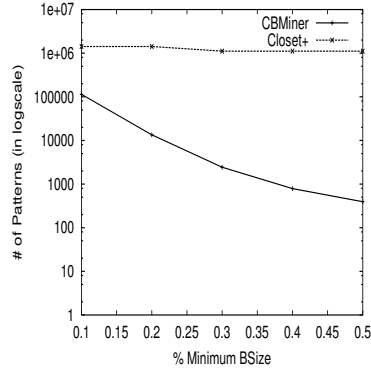
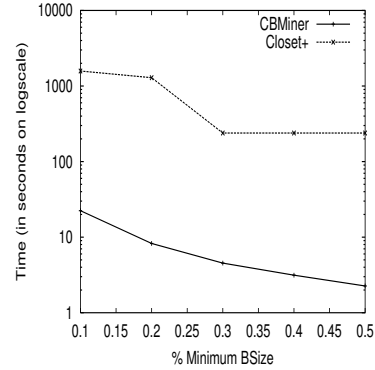
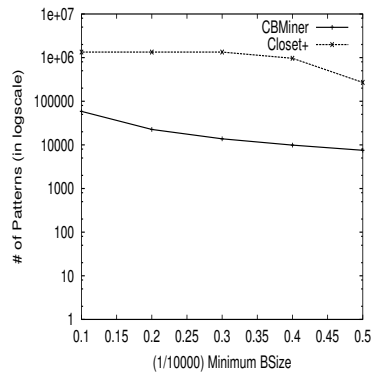
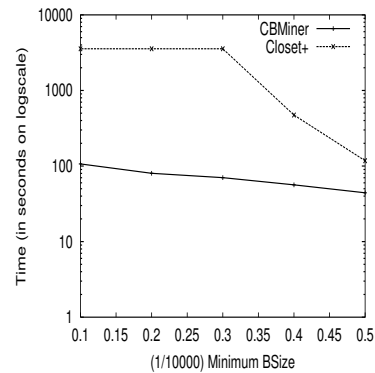
We performed numerous experiments to compare CBMINER with CLOSET+. Figs. 4–11 show the comparison results for the BSize constraint and datasets *gazelle*, *big-market*, *pumsb**, and *T40I10D100K*, while Figs. 12–13 show the results for the BSum constraint and the *sports* dataset, and Figs. 14–15 show the results for the BSim

Fig. 2. # Patt. (*big-market*).Fig. 3. Runtime (*big-market*).

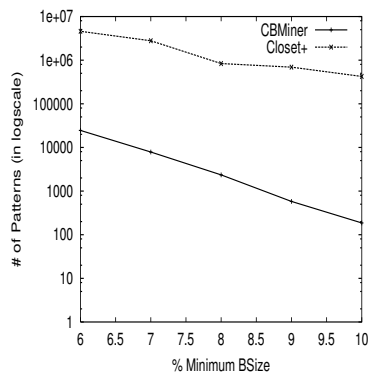
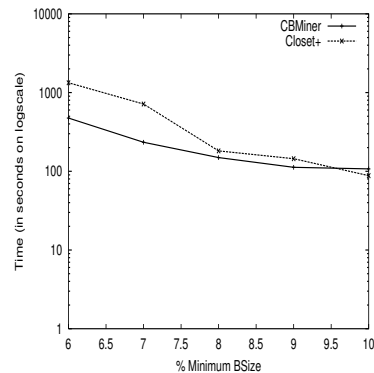
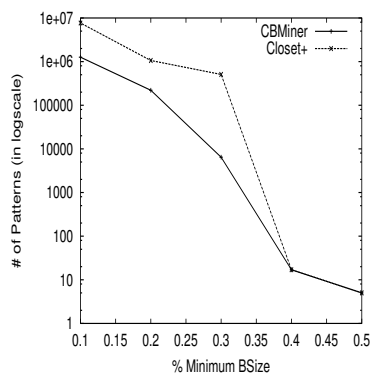
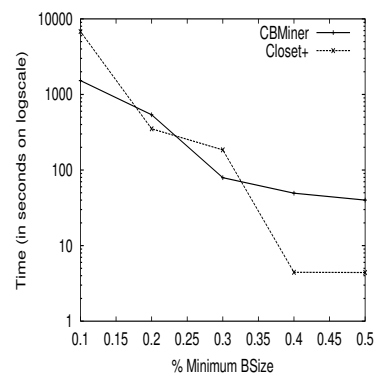
constraint and the *BMSWebView2* dataset. From these results we can see that, in general, CBMINER is substantially faster than CLOSET+. This is primary due to the fact that, as it was expected, CLOSET+ produces significantly more patterns than those produced by CBMINER. For datasets with short transactions like *gazelle*, *BMSWebView2*, and *big-market*, CBMINER can be order(s) of magnitude faster than CLOSET+, and finds order(s) of magnitude fewer patterns. While for the datasets with long transactions like *pumsb**, *sports*, and *T40I10D100K*, CLOSET+ is a little faster at high block threshold of BSize and BSum, but once the threshold is lowered, there is an explosive increase in the number of frequent closed itemsets (e.g., with BSize/BSum 0.2% CLOSET+ generates several orders of magnitude more patterns than CBMINER). These results illustrate that the pruning methods used by CBMINER are indeed effective in reducing the overall search space, leading to substantial performance improvements.

Effectiveness of the Pruning Methods

We evaluated the effectiveness of the three newly proposed pruning methods, *Column Pruning (CP)*, *Row Pruning (RP)*, and *Matrix Pruning (MP)* through their various combinations. Figs. 16–17 show the comparison results for the BSize constraint and datasets *gazelle* and *BMSWebView2*, Fig. 18 shows the results for the BSum constraint

Fig. 4. # Patt. (*gazelle*).Fig. 5. Runtime (*gazelle*).Fig. 6. # Patt. (*big-market*).Fig. 7. Runtime (*big-market*).

and the dataset *pumsb**, while Fig. 19 shows the results for the BSim constraint and the dataset *big-market*. These results show that the combination of all the three pruning techniques (denoted as CP+RP+MP) is always faster than each of the individual pruning method, and for the BSize and BSum constraints the overall ranking of the pruning effectiveness among the three methods is *Column Pruning* > *Matrix Pruning* > *Row Pruning*, while for the BSim constraint *Matrix Pruning* is more effective than *Row Pruning* and *Column Pruning*. Note that if we do not apply any of these three pruning methods (denoted as *No-Pruning*), CBMINER will run too slow, and for this reason we do not show the curves corresponding to *No-Pruning* in Figs. 14–17.

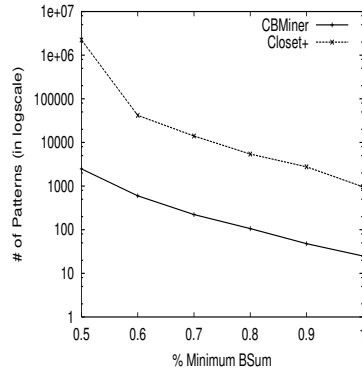
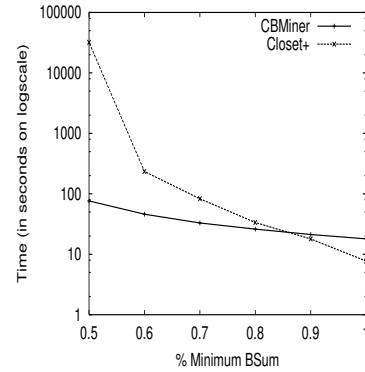
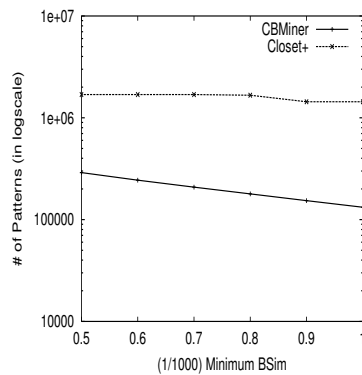
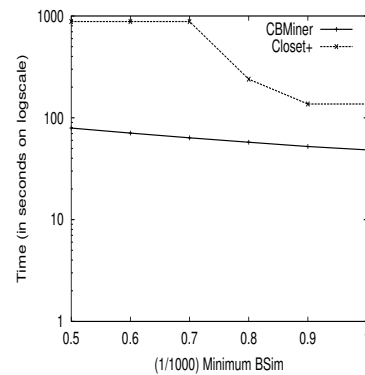
Fig. 8. # Patt. (*pumsb**).Fig. 9. Runtime (*pumsb**).Fig. 10. # Patt. (*T40I10D100K*).Fig. 11. Runtime (*T40I10D100K*).

Scalability Study

We used the IBM synthetic dataset series T10I4D x for the scalability test of CBMINER, where x indicates the base size and varies from 200K to 1000K. In the experiments we fixed the threshold for the three constraints at 0.01%. From Fig. 20, we can see that CBMINER has linear scalability on all the three constraints.

<i>Data</i>	<i>No. of documents</i>	<i>No. of terms</i>	<i>No. of classes</i>
Classic	7089	12009	4
Sports	8580	18324	7
LA1	3204	31472	6

Table 5.2: Summary of document datasets used for the application.

Fig. 12. # Patt. (*sports*).Fig. 13. Runtime (*sports*).Fig. 14. # Patt.
(*BMSWebView2*).Fig. 15. Runtime
(*BMSWebView2*).

5.1.3 Application - Micro Concept Discovery

Finally, we demonstrate an application for the three block constraints in the context of document clustering by showing that the blocks discovered by these constraints represent sets of documents that have a great chance of belonging to the same cluster and hence can be used to identify potential cores of natural clusters in data as well as thematically related words. For this application we chose two additional document datasets viz., *LA1* and *Classic* in addition to *Sports*. The *LA1* dataset contains articles that appeared in LA Times news, whereas the *Classic* dataset contains abstracts of technical papers. Some of the characteristics of these datasets are shown

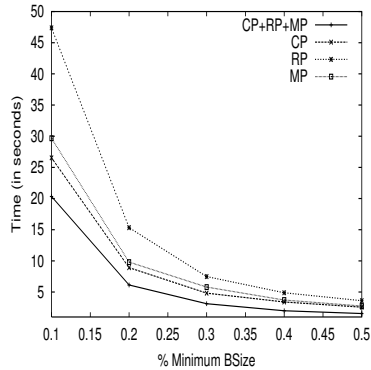


Fig. 16. Pruning methods.
(*gazelle*).

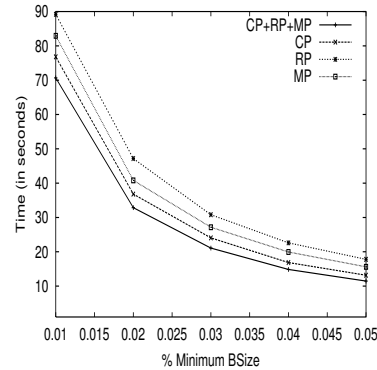


Fig. 17. Pruning methods.
(*BMSWebView2*).

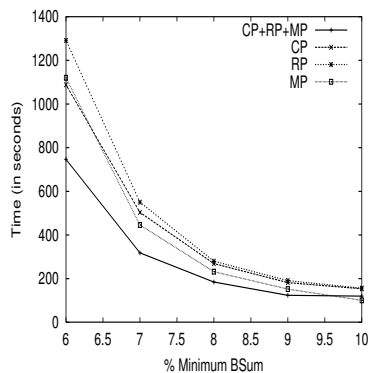


Fig. 18. Pruning methods.
(*pumsb**).

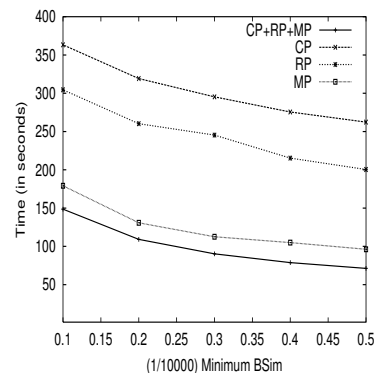


Fig. 19. Pruning methods.
(*big-market*).

in Table 5.2. We scaled the document vectors using the well known *tf-idf* scaling and normalized using L_2 -norm and used our closed block mining algorithm with block size, block-sum and block-similarity constraints. From the patterns that were found we chose the 1000 highest ranked patterns on the basis of the constraint value. For example, for the block sum constraint, we selected the top-1000 blocks ranked on block sum and in the same way for block-size and block-similarity constraints. For each of the top-1000 blocks we computed the entropies [34] of the documents that formed the supporting set of the block and took the average of the 1000 entropies. Similarly, we computed the average block pattern frequency and average block pat-

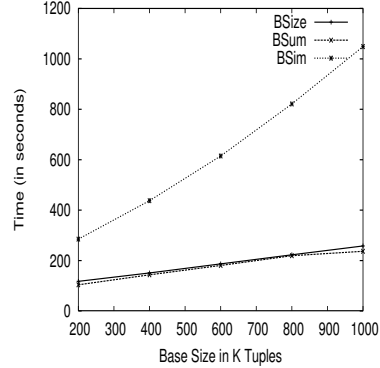


Fig. 20. scalability ($T10I4Dx$).

tern length. For comparison purposes, we also used the CLOSET+ algorithm to find a set of closed frequent itemsets and also selected the 1000 most frequent itemsets discovered by CLOSET+. Fig. 21 shows the average entropy, frequency, and length of the various patterns discovered by the four algorithms for the three datasets. Note that the CLOSET+ results are labeled as “freq”.

From these results we can see that the average entropy of the patterns discovered by the four schemes are quite small, indicating that all of them do reasonably well in identifying itemsets whose supporting documents are primarily from a single class. Despite that, we can see that the block-similarity constraint outperforms the rest, as it leads to the lowest entropies (i.e., purest clusters) for all datasets. This verifies our initial motivation for defining the block-similarity constraint, as it is able to better capture the characteristics of the underlying datasets and problem, and discover sets of words that are thematically very related. The block-size and the itemset support constraints show some inconsistency in finding good concepts as they do not account for the weights associated with the terms in the document-term matrices. On the other hand the block-sum constraint does reasonably well as it was able to take into account the differences in the terms weights provided by the $L2$ -norm and tf - idf scaling for the document vectors. Also note that the highest ranked patterns discovered by the frequent closed mining algorithm (CLOSET+) are in general quite short compared to the length of the patterns discovered by the block constraints.

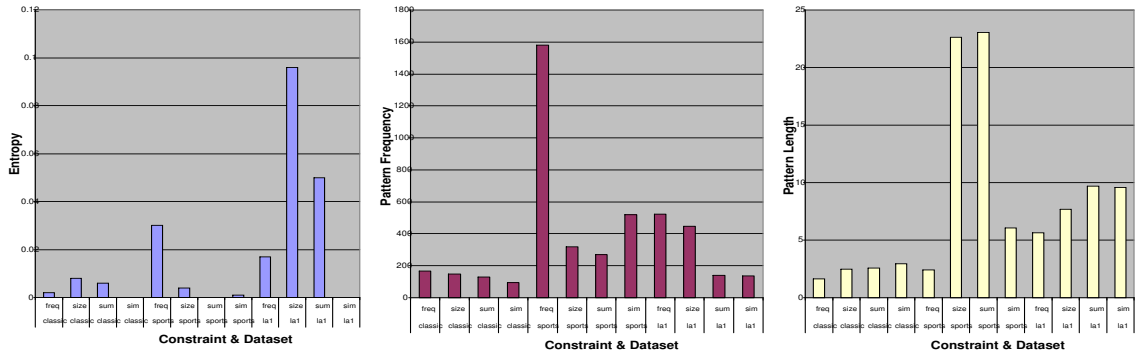


Fig. 21 Evaluation of the quality of the top-1000 patterns discovered by various algorithms.

5.2 Conclusion and Future Work

In this paper we studied how to mine valid closed patterns with tough block constraints and proposed a matrix-projection based framework called CBMINER for mining closed block patterns in transaction-item or document-term matrices effectively. Under this framework we mainly discussed three typical block constraints viz., *block size*, *block sum* and *block similarity*. Some widely adopted properties derived from the anti-monotone or monotone constraints no longer hold to be used to prune search space for these tough block constraints. As a result, we specifically proposed three novel pruning methods, *column pruning*, *row pruning* and *matrix pruning*, which can push deeply the block constraints into pattern discovery and prune the unpromising columns, rows, and projected matrices effectively. Moreover, our experimental results show CBMINER finds much fewer patterns and runs order(s) of magnitude faster than the traditional frequent closed pattern mining algorithms.

BIBLIOGRAPHY

- [1] R. Agarwal, C. Aggarwal, V. Prasad, and V. Crestana. A tree projection algorithm for generation of large itemsets for association rules. *IBM Research Report*, RC21341, November 1998.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB*, September 1994.
- [3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
- [4] R. Bayardo, R. Agrawal, and D. Gunopulos. Constrained based rule mining for large dense databases. In *Proc. of the ICDE'99*, Mar. 1999.
- [5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *ACM SIGMOD 1997*, 05 1997.
- [6] C. Bucila, J. Gehrke, D. Kifer, and W. White. Dualminer : A dual-pruning algorithm for itemsets with constraints. In *ACM KDD*, 2002.
- [7] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proc. of the ICDE 01*, April 2001.
- [8] C.K.-S.Leung, L.V.S.Lakshmanan, and R.T.Ng. Exploiting succinct constraints using fp-trees. In *ACM SIGKDD Explorations, Volume 4*, pages 40–50, 2002.
- [9] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing: Design and Analysis of Algorithms, 2nd Edition*. Addison Wesley Publishing Company, 2003.
- [10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD*, pages 1–12, May 2000.
- [11] J.Pei and J.Han. Constrained frequent pattern mining : A pattern-growth view. In *ACM SIGKDD Explorations, Volume 4*, pages 31–40, 2002.
- [12] J.Pei, J.Han, and L.V.S.Lakshmanan. Mining frequent itemsets with convertible constraints. In *IEEE ICDE*, 2001.

- [13] G. Karypis and E. H. Han. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *Proc. of CIKM*, 2000.
- [14] D. Kifer, C. Bucila, J. Gehrke, and W. White. How to quickly find a witness. In *22nd ACM SIGACT-SIGMOND-SIGART Symposium on Principles of Database Systems (PODS)*, 2003.
- [15] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *Knowledge Discovery and Data Mining*, pages 337–341, 1999.
- [16] G. Liu, H. Lu, W. Lou, and J.X. Yu. On computing and querying frequent patterns. In *Proc. of the ACM SIGKDD'03*, Aug. 2003.
- [17] M.Zaki and C.Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proc. SDM'02*, April 2002.
- [18] R. Ng, Laks V. S. Lakshmanan, J. Han, and T. Mah. Exploratory mining via constrained frequent set queries. In *Proc. of the ACM SIGMOD'99*, June 1999.
- [19] Jing Soo Park, M. Chen, and Philip S. Yu. An effective hash based algorithm for mining association rules. In *Proc. of the ACM SIGMOD'95*, 1995.
- [20] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the ICDT 99*, Jan. 1999.
- [21] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proc. of ACM SIGKDD*, 2000.
- [22] J. Pei, J. Han, H. Liu, and S. Nishio et al. H-mine : Hyper structure mining of frequent patterns databases. In *IEEE Conference on Data Mining*, 2001.
- [23] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 of ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery*, 2000.
- [24] Jr. R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. of the ACM SIGMOD'98*, June 1998.
- [25] R.Srikant, Q.Vu, and R.Agrawal. Mining associations rules with item constraints. In *3th ACM SIGKDD*, pages 67–73, 1997.

- [26] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.
- [27] M. Seno and G. Karypis. Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *ICDM*, 2001.
- [28] M. Seno and G. Karypis. Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *ICDM*, 2002.
- [29] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. of the ACM SIGKDD'03*, Aug. 2003.
- [30] J. Wang and G. Karypis. Bamboo: Itemset mining by deeply pushing the length-decreasing support constraint. Technical Report TR #03-40, Department of Computer Science, University of Minnesota, Minneapolis, MN, 2003. Available on the WWW at <http://cs.umn.edu/~karypis/publications>.
- [31] K. Wang, Y. Jiang, J. Xu Yu, G. Dong, and J. Han. Pusing aggregate constraints by divide-and-approximate. In *Proc. of ICDE*, 2003.
- [32] Ke Wang, Yu He, and Jiawei Han. Mining frequent itemsets using support constraints. In *The VLDB Journal*, pages 43–52, 2000.
- [33] M. Zaki. Generating non-redundant association rules. In *6th ACM SIGKDD*, pages 34–43, 2000.
- [34] Ying Zhao and George Karypis. Criterion functions for document clustering: Experiments and analysis. *Machine Learning*, in press.