Parallel Frequent Item Set Mining with Selective Item Replication

Eray Özkural, Bora Uçar, and Cevdet Aykanat

Abstract—We introduce a transaction database distribution scheme that divides the frequent item set mining task in a top-down fashion. Our method operates on a graph where vertices correspond to frequent items and edges correspond to frequent item sets of size two. We show that partitioning this graph by a vertex separator is sufficient to decide a distribution of the items such that the subdatabases determined by the item distribution can be mined independently. This distribution entails an amount of data replication, which may be reduced by setting appropriate weights to vertices. The data distribution scheme is used in the design of two new parallel frequent item set mining algorithms. Both algorithms replicate the items that correspond to the separator. NoClique replicates the work induced by the separator and NoClique2 computes the same work collectively. Computational load balancing and minimization of redundant or collective work may be achieved by assigning appropriate load estimates to vertices. The experiments show favorable speedups on a system with small-to-medium number of processors for synthetic and real-world databases.

Index Terms—Parallel data mining, frequent item set mining, mining methods and algorithms, selective data replication, graph partitioning by vertex separator.

1 Introduction

1.1 Frequent Item Set Mining Problem

A transaction database consists of a multiset $T = \{X|X \subseteq I\}$ of transactions. Each transaction is an item set, and it is drawn from a set I of all items. In practice, the number of items, $|I|$, is in the order of magnitude of 10^3 or more. The number of transactions, $|T|$, is usually larger than 10^5.[1] A pattern (or item set) is $X \subseteq I$, any subset of I, while the set of all patterns is 2^I. The frequency function $f(T, x) = |\{X \in T|x \in X\}|$ computes the number of times a given item $x \in I$ occurs in the transaction database T, and it is extended to item sets as $f(T, X) = |\{Y \in T|X \subseteq Y\}|$ to compute the frequency of a pattern. We use just $f(x)$ or $f(X)$ when T is clear from the context.

Frequent item set mining (FIM) is the discovery of patterns in a transaction database with a frequency of support threshold ϵ and more. The set of all frequent patterns is $\mathcal{F}(T, \epsilon) = \{X \in 2^I|f(T, X) \geq \epsilon\}$. We use just $\mathcal{F}$ when T and ϵ are clear from the context. In our algorithms, two sets require special consideration. $F = \{x \in I|f(T, x) \geq \epsilon\}$ is the set of frequent items, and $F_2 = \{X \in \mathcal{F}||X| = 2\}$ is the set of frequent patterns with cardinality 2. In general, F_k is the set of frequent patterns with cardinality k. A significant property of FIM known as downward closure states that subsets of a frequent pattern are frequent, i.e., if $X \in \mathcal{F}(T, \epsilon)$ then $\forall Z \subset X, Z \in \mathcal{F}(T, \epsilon)$ [1].

1. These numbers come from the parameters used for the synthetic data generator in [1].

- *E. Özkural and C. Aykanat are with Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey. E-mail: {erayo, aykanat}cs.bilkent.edu.tr.*
- *B. Uçar is with LIP, ENS Lyon, 69364 Lyon, France. E-mail: bora.ucar@ens-lyon.fr.*

Manuscript received 24 Dec. 2009; revised 7 July 2010; accepted 14 Oct. 2010; published online 18 Jan. 2011.
Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-12-0666.
Digital Object Identifier no. 10.1109/TPDS.2011.32.

If all item sets in $\mathcal{F}$ are enumerated, the problem is known as the all FIM problem. Since the size of $\mathcal{F}$ can be large, smaller enumeration problems have been defined such as closed [2] and maximal [3] FIM problems.

1.2 Related Work and Motivation

FIM comprises the core of several data mining algorithms, such as association rule mining and sequence mining. Frequent pattern discovery usually dominates the running time of these algorithms, therefore much research has been devoted to increasing the efficiency of this task. Since both the data size and the computational costs are large, parallel algorithms have been studied extensively [4], [5], [6], [7], [8], [9], [10], [11], [12]. FIM has become a challenge for parallel computing since it is a complex operation on huge databases requiring efficient and scalable algorithms.

While there are a host of advanced algorithms for parallel FIM, it is desirable to achieve better flexibility and efficiency. We have been inspired by the *Partition* algorithm [13] which divides the database horizontally and merges individual results, as well as Zaki et al.'s *Par-Eclat* algorithm [5] which redistributes the database into parts that can be mined independently. Also of immediate interest are the parallelizations of *Apriori* [1], most notably *Candidate-Distribution* [4] which pioneered independent mining. We ask the following questions. Can we design a parallel algorithm that exploits data-parallelism and task-parallelism? Can we find a model to optimize its performance? The present paper gives an affirmative answer to these questions by introducing an algorithm that divides the database into independently mined parts in a top-down fashion, according to an optimized distribution of the item set.

A review of related work with emphasis on parallelizations of *Apriori* and *Par-Eclat* may be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.32.

1.3 Contributions

We introduce two new coarse-grain data-parallel FIM algorithms using a top-down data partitioning scheme with selective replication. We propose a novel divide-and-conquer strategy suitable for parallelization of the FIM task. Our objective is to divide the whole transaction database into parts that can be mined independently. It turns out that we can distribute items so as to achieve our goal of independent mining, while replicating some items selectively, implying an amount of work that cannot be divided further in the same fashion. This optimization problem is cast as a Graph Partitioning by Vertex Separator (GPVS) problem where the partitioning objective corresponds to minimizing data replication or collective work (work that requires collective communication) by setting appropriate weights to vertices, and the partitioning constraint corresponds to maintaining storage balance or computational load by setting appropriate weights likewise. The transaction database distribution is independent of the underlying database representation and the serial mining algorithms employed. Experiments show that our method has competitive performance with respect to a state-of-the-art parallel mining implementation.

2 Transaction Database Distribution

In this section, we describe our theoretical contributions which will be developed into a parallel algorithm in Section 3. We make heavy use of the GPVS problem, which is briefly explained in the following.

The GPVS problem is to find a minimum weighted vertex separator V_s, removal of which decomposes a graph into components with roughly equal weights [14]. Let $G = (V, E)$ be a graph where $w(u)$ is the weight of vertex u. Let $w(U) = \sum_{u \in U} w(u)$ be the weight of a vertex set U. Let $Adj(u)$ denote the set of vertices that are adjacent to u, i.e., $Adj(u) = \{v | (u, v) \in E\}$. This operator can be extended to vertex sets by letting $Adj(U) = \bigcup_{u \in U} Adj(u) - U$.

Definition 1 (n-way GPVS). *$\Pi_{VS}(G) = \{V_1, V_2, \ldots, V_n : V_s\}$ is a partition of the vertex set V into $n + 1$ subsets $V_1, V_2, \ldots, V_n$ and V_s such that for all $1 \leq i < j \leq n$ $Adj(V_i) \cap V_j = \emptyset$ (i.e., $Adj(V_i) \subseteq V_s$). The partitioning objective is to minimize $w(V_s)$. The partitioning constraint is, for all $1 \leq i \leq n$, $w(V_i) \cong [w(V) - w(V_s)]/n$ (parts have roughly the same weight).*

The problem is NP-complete [15, ND 25 Minimum b-vertex separator]. A separator V_s is said to be minimal if there is no subset of V_s that is also a separator. The two-way GPVS will be denoted as $\Pi_{VS}(G) = \{A, B : S\}$.

We introduce a distribution method that can be used to divide the FIM task in a top-down fashion. The method operates on the graph G_{F_2} which is defined as follows:

Definition 2. *$G_{F_2}(T, \epsilon) = (F, F_2)$ is an undirected graph in which each vertex $u \in F$ is a frequent item and each edge $\{u, v\} \in F_2$ is a frequent pattern of length two, for a given database T and support threshold ϵ. The parameters T and ϵ will be dropped when they are clear from the context.*

We decode a two-way GPVS of the G_{F_2} graph as a two-way distribution of the transaction database such that the two subdatabases obtained can be mined independently and therefore utilized for concurrency. In order for this property to hold, there is an amount of replication dictated by the vertex separator of G_{F_2}, which corresponds to the partitioning objective of GPVS. In the following, we first present the optimization aspects of our transaction database distribution technique. Then, we expound on our GPVS model for two-way transaction database distribution. Afterwards, we discuss minimization of data replication, followed by minimization of collective work and load balancing in the GPVS model. We then extend the two-way distribution scheme to n-way (for n processors). Last, we show that our method is applicable to maximal and closed FIM problems.

2.1 Optimizing Parallel Frequent Item Set Discovery

Our objective of transaction database distribution is to divide a transaction database such that each subdatabase can be mined independently, while not inflating the data prohibitively and keeping the computational load balanced across subdatabases. Once such a distribution is obtained, a coarse-grain parallel frequent item set mining algorithm similar to *Par-Eclat* can be designed. *Par-Eclat* consists of a redistribution phase and a following local mining phase with no communication [5]. We present two algorithms: *NoClique* features completely independent mining with no communication just like *Par-Eclat*, while *NoClique2* has a collective phase in which the running time is minimized and the rest of mining is independent. Since some data mining tasks on the subdatabases are performed independently in either algorithm, our method may be classified as a data-parallel algorithm that adopts input data partitioning with replication. This input data partitioning induces a task partitioning according to the owner-computes rule [16, Section 3.2.2], which states that the process assigned a particular data item is responsible for all computation associated with it.

We show that GPVS on G_{F_2} is sufficient to designate such a distribution on the transaction database. Our work assumes that G_{F_2} is sparse, because GPVS may not be feasible on dense graphs. Note that a sparse G_{F_2} does not necessarily require the input database to be sparse.

We may begin formulating a problem for the coarse-grain data-parallel frequent item set mining algorithm as follows: Let a database T contain a smaller database T_i. T_i is a subdatabase of database T if and only if, for every transaction $X \in T_i$, there is a distinct transaction $Y \in T$ such that $X \subseteq Y$ (recall that T and T_i are multisets). We will denote this ordering relation with $T_i \prec T$. The input database T is distributed to a number of processors such that each processor has a subdatabase of the original transaction database. We denote this distribution by $D(T) = \{T_i | T_i \prec T\}$, possibly with replication. Also, we require the union of frequent patterns discovered in individual processors to be the set of frequent patterns of the entire data, i.e., $\mathcal{F}(T, \epsilon) = \bigcup_{T_i \in D(T)} \mathcal{F}(T_i, \epsilon)$. We call this the independent mining condition for a distribution $D(T)$.

In the following optimization problem, $w(\cdot)$ is any sensible cost measure that relates to mining a database, e.g., computational work, data size:

$$\text{minimize}\left(\sum_{T_i \in D(T)} w(T_i)\right) - w(T), \quad (1)$$

$$\text{subject to } T_i \prec T, \text{ for all } T_i \in D(T), \quad (2)$$

$$\mathcal{F}(T, \epsilon) = \bigcup_{T_i \in D(T)} \mathcal{F}(T_i, \epsilon), \quad (3)$$

$$w(T_i)\text{'s are approximately equal.} \quad (4)$$

The objective in (1) seeks to minimize the total amount of redundancy that the distribution $D(T)$ entails. We subtract the cost of the entire database from the sum of costs of distributed subdatabases T_is to denote this. Equation (2) is the distribution condition which states that the transaction database is distributed in any fashion, e.g., transactionwise, itemwise, or hybrid. Equation (3) is the independent mining condition, which ensures that independent mining of the subdatabases yields the frequent patterns of the entire database. The balancing condition (4) ensures that all processors share the cost fairly. At this stage, we do not explicitly state whether we are minimizing data redundancy or parallel overhead. However, some amount of data replication is often necessary for the independent mining condition to hold.

We will now expose our particular item redistribution scheme using information in frequent item sets of length two, which can be easily computed in parallel like in the design of *Par-Eclat* [5]. First, we will show how we can satisfy the distribution and independent mining conditions by showing a two-way item distribution. We will then analyze the objective and the balancing condition, explaining how we can assign weights and achieve load balance so that it becomes an acceptable solution to the coarse-grain parallel FIM problem.

2.2 Two-Way Itemwise Transaction Database Distribution

G_{F_2} is relatively easy to compute with respect to the complexity of the whole mining task, and its computation is amenable to efficient parallelization. It contains information that can be used to predict computational properties. For instance, the maximal cliques in G_{F_2} give us potentially maximal patterns [5], which in turn can be used to achieve task parallelism. Our data decomposition method, on the other hand, does not require finding maximal cliques. Instead, we use the GPVS of G_{F_2}, which allows us to define independent mining on the transaction database by finding a particular distribution of the item set I. Our item distribution identifies the absence of cliques across two sets of items rather than enumerating all cliques as in [5].

We will start by observing the similarity of GPVS objectives to ours. It turns out that we can use a GPVS of G_{F_2} to satisfy the independent mining conditions and to optimize parallelism at the same time. FIM task can be decomposed into mining two itemwise projections of the transaction database using GPVS. We use the projection operator π to explicitly show the vertical projections.

Definition 3. *A transaction database projected from T over a set of items X is $\pi_X(T) = \{Y \cap X | Y \in T\}$ where Y is a transaction in T.*

Recall that a two-way GPVS is denoted as $\Pi_{VS}(G) = \{A, B : S\}$ where S is the vertex separator; and A and B are vertex parts. GPVS of G_{F_2} corresponds to a certain two-way distribution $\{A \cup S, B \cup S\}$ of the item set I. This distribution induces a two-way transaction set distribution as follows:

Transaction	a	b	c	d	e	f	g	h	i
$t_1 = \{b, c, f, g\}$		×	×			×	×		
$t_2 = \{c, g\}$			×				×		
$t_3 = \{b, h, i\}$		×						×	×
$t_4 = \{b, e\}$		×			×				
$t_5 = \{a, d, g, h\}$	×			×			×	×	
$t_6 = \{d, e\}$				×	×				
$t_7 = \{b, c, e, f\}$		×	×		×	×			
$t_8 = \{a, b, c, f\}$	×	×	×			×			
$t_9 = \{b, c, d, h, i\}$		×	×	×				×	×
$t_{10} = \{b, c, e, g\}$		×	×		×		×		
$t_{11} = \{a, d, e, g\}$	×			×	×		×		
$t_{12} = \{d, h\}$				×				×	
$t_{13} = \{a, d, e, h\}$	×			×	×			×	
$t_{14} = \{b, e\}$		×			×				
$t_{15} = \{a, e, g\}$	×				×		×		

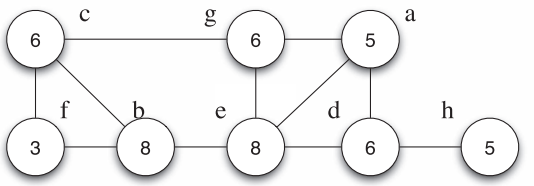


Fig. 1. (a) A sample database T with 15 transactions and 9 items. (b) G_{F_2} graph of T with a support threshold of 3. The vertices are labeled with the number of times an item occurs in the database.

Definition 4. *A two-way transaction database distribution $D(T) = \{T_1, T_2\}$ is induced by $\Pi_{VS}(G_{F_2}) = \{A, B : S\}$, where $T_1 = \pi_{A \cup S}(T)$ and $T_2 = \pi_{B \cup S}(T)$.*

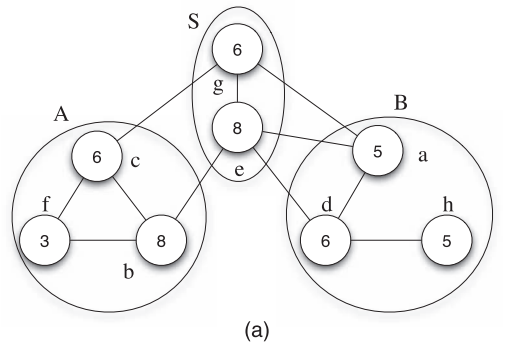
We require S to be a minimal separator. If S were not minimal, since the cost induced by the separator is included in both projections, removing a vertex from the separator would decrease the parallel cost. For that reason, it is better to choose a minimal separator, in case the GPVS heuristic does not find one. Fig. 1 depicts a sample transaction database and its G_{F_2} graph. Π_{VS} of this graph and the transaction database distribution $D(T)$ induced by Π_{VS} is illustrated in Fig. 2. In the following text, we show that mining the database parts separately results in complete FIM of the original transaction database T satisfying (3). The proofs are found in Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.32.

Lemma 1. *If there is a frequent pattern P in T, then there is a corresponding clique in G_{F_2}, with vertices corresponding to items in P.*

Lemma 2 (NoClique). *There is no frequent pattern with items in both A and B parts of $\Pi_{VS} = \{A, B : S\}$ of G_{F_2}.*

Theorem 1 (Independent Mining). *Independent discovery of frequent patterns in projected databases $T_1 = \pi_{A \cup S}(T)$ and $T_2 = \pi_{B \cup S}(T)$ results in discovery of all frequent patterns in T.*

Theorem 1 can be improved slightly to suggest a more efficient parallelization. The frequent item sets within S do not have to be mined redundantly.

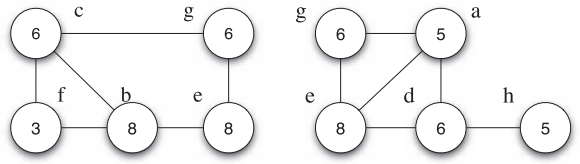


(a)

b	c	e	f	g
×	×		×	×
	×			×
×				
×		×		
				×
		×		
×	×	×	×	
×	×		×	
×	×			
×	×	×		×
		×		×
		×		×
×		×		
		×		×

a	d	e	g	h
			×	
			×	
				×
		×		
×	×		×	×
	×	×		
		×		
×				
	×			×
		×	×	
×	×	×	×	
	×			×
×	×	×		×
	×			
×	×			

(b)



(c)

Fig. 2. (a) A GPVS of the G_{F_2} graph of Fig. 1. Parts A, B, and separator S are shown. (b) Distribution $D(T) = (T_1, T_2)$ of transaction database. (c) The G_{F_2} graphs of T_1 and T_2.

Corollary 1 (Collective Work). *Consider the partition of items in Theorem 1. All patterns in T can be mined by mining frequent item sets that fall within S, independently mining A and B, and then extending frequent item sets within S by those item sets mined within A and B.*

2.3 Minimizing Data Replication

Data replication in distribution $D(T) = \{T_1, T_2\}$ is determined by the vertex separator S. By definition of the two-way distribution, for every transaction $X \in T$, $X \cap S$ is projected in both T_1 and T_2.

Lemma 3. *The amount of data replication in two-way transaction database distribution $D(T)$ given in Definition 4 is equal to $\sum_{u \in S} f(u)$.*

The amount of data replication is related to the sparsity of G_{F_2} graph. We expect the replication to grow rapidly beyond a certain edge density that is determined by the support threshold.

Lemma 4 (Minimum Replication). *GPVS of G_{F_2} with item frequencies as vertex weights minimizes the amount of data replication.*

Minimizing data replication is also correlated to minimizing the total volume of communication during database redistribution. If the database is to be provided from a central server, then both objective functions are identical. Moreover, for an initial random distribution of the database, we are minimizing the upper bound of total communication volume during the redistribution phase. Note that the GPVS model will maintain storage balance among processors due to the partitioning constraint.

2.4 Minimizing Collective Work

Here, we take a look at possible choices for $w(\cdot)$ to minimize collective work. If the computational work estimate for a projection over a set of items X is in the form of a summation of individual load estimates $l(\cdot)$ for items:

$$w(\pi_X(T)) = \sum_{u \in X} l(u), \tag{5}$$

then the proposed GPVS model will minimize collective work instead of minimizing data replication. It will also balance computational load due to the partitioning constraint.

Estimating the computational load is nontrivial, since we cannot know in advance how many patterns are present in the data. However, we can reason about the potential number of item sets in the search space that the mining algorithm will need to traverse. Although every algorithm follows a different strategy for determining frequent patterns, a measure of the portion of the search space containing potentially frequent patterns gives us a good estimate as in [4], [5]. In our method, however, computing the maximal cliques in G_{F_2} (like in [5]) will incur additional overhead. Therefore, we use simpler functions for load estimation such as the following:

$$w_1(\pi_X(T)) = \sum_{u \in X} f(u), \tag{6}$$

$$w_2(\pi_X(T)) = \sum_{u \in X} \binom{d(u)}{2}, \tag{7}$$

$$w_3(\pi_X(T)) = \frac{1}{2} \sum_{(u,v) \in X^2} f(\{u, v\}). \tag{8}$$

For estimating computation time, we can use (6) which calculates the data size within the projection over a given item set X in a fashion resembling [4]. Equation (7) does not take into account the actual complexity of the task. An alternative approximation, which is inexpensive, can be found in [5]. Equation (7) is based on Zaki et al.'s item set clustering [5] where $d(u)$ is the degree of vertex u in G_{F_2}. This estimate is an upper bound on the number of potential frequent patterns of length 3 obtained by calculating the number of 2-combinations of patterns with length 2. Naturally, more advanced load estimate methods can be used to improve the accuracy. An obvious choice among the simpler functions is the total frequency of G_{F_2} edges that fall within a given item set X which gives us (8). Although $w_3(\cdot)$ does not strictly conform to (5), it can be made so by evenly

distributing the weight of each edge among its incident vertices, which yields an approximation to (8). In our experiments, we have found that $w_1(\cdot)$ performed better or as well as $w_2(\cdot)$ and $w_3(\cdot)$ perhaps because it tends to reduce both data and task overhead.

2.5 Extension to n-Way Distribution and Any Level k of Mining

We will now show means to extend two-way transaction database distribution to an n-way distribution $D(T) = \{T_1, T_2, \ldots, T_n\}$, where the independent mining conditions are generalized in the obvious way. The two-way transaction database distribution can be applied recursively to divide the two projected databases. Since the resulting projected databases are transaction databases themselves, we can apply the same method to divide them further.

In order to distribute the derived databases, one must obtain the G_{F_2} of the two parts. This can be accomplished by simply running the same algorithm for the projected transaction database, however, this can be costly. In the following, we present facts that lead to an efficient computational scheme to calculate an n-way distribution directly over G_{F_2}. By making use of this simple observation, we avoid constructing intermediate projected databases. There is no need to recompute F and G_{F_2}, since they are already known as shown by the following lemma:

Lemma 5 (G_{F_2} of a Projection). *For a given item set $X \subseteq I$, $G_{F_2}(\pi_X(T), \epsilon)$ is the subgraph of $G_{F_2}(T, \epsilon)$ induced by the vertex set X.*

We thus observe that we do not need to construct intermediate databases to calculate the G_{F_2}s of the sub-databases in $D(T)$.

Corollary 2 (Fast Recursive Distribution). *Regarding the distribution $D(T) = \{\pi_{A\cup S}(T), \pi_{B\cup S}(T)\}$ induced by $\Pi_{VS}(G_{F_2}) = \{A, B : S\}$, the $G_{F_2}(\pi_{A\cup S}(T), \epsilon)$ and $G_{F_2}(\pi_{B\cup S}(T), \epsilon)$ can be calculated as vertex-induced subgraphs of $G_{F_2}(T, \epsilon)$ by vertex sets $A \cup S$ and $B \cup S$, respectively.*

The simplest way to obtain an n-way distribution is to use an n-way GPVS directly. Independent mining results extend to the n-way case in an obvious fashion. Thus, we will not prove them separately. However, there are a few differences from the two-way case, which we will now portray. In an n-way GPVS $\Pi_{VS}(G_{F_2}) = \{V_1, V_2, \ldots, V_n : S\}$ of the G_{F_2} graph, we note that the projection of $S \cup V_i$ will result in independent mining. Although S is a minimal separator (i.e., no subset of it is a separator), we observe that not all S need to be replicated in all parts. In general, a portion of S will have to be replicated on processor i (i.e., $Adj(V_i) \cap S$) which may in the worst case correspond to S. This implies that an item in S may be replicated in a different number of projected databases than others in the resulting distribution. The n-way GPVS model does not encapsulate this fact. However, it is easier to implement with an n-way GPVS tool.

Our formulation is also applicable to levels higher than two in case G_{F_2} is too dense. We define a graph G_{F_k} of k-length frequent item sets as follows:

Definition 5. *$G_{F_k}(T, \epsilon) = (F, E)$ is an undirected graph in which each vertex $u \in F$ is a frequent item. For each frequent item set X of length k in F_k, we insert a clique of items in X into this graph, i.e., one edge for each length 2 support of X.*

This definition allows us to use all the relevant results with no modification. The extension of results is trivial and will not be detailed due to space considerations. However, one property is important:

Lemma 6 (Sparsity of Higher Levels). *$G_{F_{k+1}}$ is not denser than G_{F_k}.*

2.6 Maximal and Closed FIM Problems

Our method is applicable to both variations of the FIM problem that compute subsets of $\mathcal{F}$. In maximal FIM, no set that is a subset of a frequent item set is output [3], [17]. In closed FIM, no set that is a subset of a frequent item set and is supported by the same transactions is output [2]. For instance, consider frequent item set $X = \{a, b, c\}$. In maximal FIM, no subset of X like $\{b, c\}$ will be output, and in closed FIM, $\{b, c\}$ will be output if and only if it occurs in a different set of transactions than X. After item distribution, if a processor has a set of frequent items X, it also has all transactions belonging to all subsets of X. Thus, both maximal and closed item set mining can be parallelized with our method.

3 Two Data-Parallel Algorithms

In this section, we present *NoClique* and *NoClique2*, which are coarse-grain data-parallel algorithms based on the theoretical observations of Section 2. Our algorithms compute the set of frequent item sets and their frequencies for a given global transaction database T and a support threshold ϵ on n processors. The implementation of *NoClique2* is built upon our new vertical serial FIM algorithm *Bitdrill*.

3.1 NoClique: The Black Box Parallelization

NoClique is a direct application of Theorem 1 and Corollary 2. First, we compute G_{F_2}. Then, we recursively apply the two-way database distribution of Definition 4 until we have n parts, using fast recursive distribution (Corollary 2). For instance, assume $n = 4$. Consider the two-level partitioning that results in the G_{F_2} graphs of T_1 and T_2 in Fig. 2. We have parts A, B, and separator S at the top level; we take two vertex-induced subgraphs of G_{F_2} over $A \cup S$ and $B \cup S$. If we apply GPVS recursively on $G_{F_2}(T_1)$ and $G_{F_2}(T_2)$, we can obtain four overlapping item sets that define an item distribution such as $D(I) = \{\{b, c, f, g\}, \{b, e, g\}, \{a, d, e, g\}, \{d, e, h\}\}$. Now, each item set in $D(I)$ can be assigned to a processor. The database is redistributed to processors according to this assignment. Afterwards, we can run any given sequential FIM algorithm on each processor simultaneously and independently, with no further communication. The main advantage of this parallelization is that any serial FIM algorithm that starts from level 3 can be used. The disadvantage is that, since some subgraphs of $G_{F_2}(T)$ are replicated, there is some redundant work. Therefore, this algorithm is suitable only for sparse problem instances that do not require much replication. The recursive application of the two-way item distribution can be carried out in parallel, and of course it is much better if a parallel GPVS algorithm

can be used. We have obtained extremely high superlinear speedups in the parallelization of *FP-Growth* and *AIM2* which prompted us to continue research in this direction. We applied *NoClique* to parallelize *kDCI* [11], [18], [19], *LCM* [20] (all FIM), *DCI-Closed* [11], *AIM* [21] (version 2), and *FP-Growth-Tiny* [22].

3.2 Bitdrill: Our Sequential Mining Algorithm

Bitdrill is a new efficient sequential FIM code that we developed as a basis for our *NoClique2* algorithm. It uses tries (prefix trees) to store sets of item sets, where each item set is a string of items in decreasing order of frequency. It uses tidlists (a tidlist is a list of transaction ids an item occurs in) to store the database in memory; linked lists of items are used for sparse items and bit vectors are used for dense items. The algorithm proceeds in BFS order and affords fast candidate generation in a fashion similar to *kDCI* (which is one of the most efficient FIM algorithms together with *LCM*). We use a regular tree data structure instead of prefix arrays as in *kDCI*. Fast candidate generation relies on the fact that the prefix tree already captures much of the proximity between two item sets needed for generating a candidate. Let A and B be two frequent item sets of length k that share a prefix of length $k-1$. Both will be the children of the same internal node in the prefix tree. Thus, one can simply take their union and generate a $(k+1)$-length candidate item set. When we consider the Downward Closure lemma, we will see that all candidates can be generated in this fashion since any subset of a candidate must be frequent and will have frequent subsets with all possible $(k-1)$-length prefixes. Thus, we can simply traverse the prefix tree and generate all candidates by taking 2-combinations of the children of each internal tree node that corresponds to a $(k-1)$-length prefix. After the candidate is generated, it is subject to further pruning employing the Downward Closure lemma. Since we use a vertical representation, the frequency of candidates can be calculated on the fly. To speed up the tidlist intersections, we use a cache to hold all the tidlist intersections in the path to the root, so that a single additional intersection is sufficient to count the transactions in a candidate item set. The overall algorithm is quite efficient; its performance is comparable to *kDCI* for dense databases and is faster than *kDCI* for sparse databases (due to the dynamic tidlist representation).

3.3 NoClique2 Algorithm

3.3.1 Assumptions

We assume that the number of items is much greater than n (the number of processors). We assume that the database has already been mined up to level l and a GPVS of G_{F_l} has been computed. In the following, we use k as a variable level and we start mining from level $l+1$. Our algorithm will work better when G_{F_l} can be partitioned well. In many cases, there is a suitable l.

3.3.2 Overview

Using our n-way GPVS-based item distribution/replication scheme, we decompose the mining problem into a collective work phase (with communication), and independent work phase (with no communication) following the observations in Corollary 1. The algorithm takes as input at each processor a local transaction database T_{local}, and an absolute support threshold ϵ. We assume that T has been partitioned transactionwise into T_{local}s prior to the execution of the mining algorithm. We also supply the set of frequent item sets up to and including level l, the graph G_{F_l} corresponding to level l, and a heuristic GPVS solution $\Pi_{VS}(G_{F_l})$. The algorithm is comprised of four phases:

1. Redistribute items with selective replication.
2. Mine replicated items in parallel.
3. Mine nonreplicated items independently.
4. Merge frequent item sets across replicated and nonreplicated sets of items.

The phases of our algorithm are explained in the following.

3.3.3 Redistribution of Items

Items are distributed according to an n-way GPVS of G_{F_l}. The items in the separator V_s are replicated on each processor. Every other part V_i in the partition contains items collected on a distinct processor. Using the notation of *NoClique*: $D(I) = \{V_i \cup V_s | V_i \in \Pi_{VS}(G_{F_l})\}$.

The horizontal input databases are scanned and using all-to-all personalized communication, each processor receives the parts of transactions that it requires according to the item distribution. After that, each processor constructs tidlists of those items.

3.3.4 Mining Replicated Items in Parallel

Since each processor has the tidlists of all the items in V_s, we can parallelize candidate generation and testing steps fairly well, starting from level $l+1$. Assume that for a previous level k, we have the frequent item sets inserted in decreasing frequency order into a prefix tree. On the prefix tree, we can efficiently generate candidates for level $k+1$ using fast candidate generation of *Bitdrill*. While traversing an internal node for a $k-1$ length prefix during fast candidate generation (Section 3.2), for a children (all of which are leaves) at most a^2 candidates can be generated. Those internal nodes are each given the just mentioned upper bound of a^2 as weight and we partition the prefix tree into n subtrees of alphanumerically consecutive item sets, where each subtree has a roughly equal sum of weights. Each processor generates a distinct set of candidates with fast candidate generation on the assigned subtree, and then intersects tidlists to check their frequencies, simultaneously. At the end of the iteration, the (locally output) frequent item sets of length $k+1$ are gathered on all processors. The iteration continues until frequent item sets are exhausted. Since both candidate generation and testing steps are parallel, and the subtree-based distribution of candidates makes local tidlist caches useful, this phase works fairly fast.

3.3.5 Independent Mining

On each processor i, there is a distinct set of tidlists corresponding to items in V_i not present on any other processor. The frequent item sets within V_i are mined using a levelwise vertical mining algorithm (*Bitdrill*) starting from level $l+1$.

3.3.6 Merging Frequent Item Sets

As the last step, we mine frequent item sets that have items in both the replicated V_s and the nonreplicated items V_i (for

a processor i). We use the output of two preceding phases to achieve this. We start with level $l+1$ again. For merging frequent item sets in a level $k+1$, assume that we have the frequent item sets in level k. We use both the frequent items in level k and the already mined frequent item sets in V_s and V_i to prune as many frequent item sets as possible. We apply the well-known Downward Closure pruning. Furthermore, any generated candidate must be combined from already-mined two sets of frequent items that we are merging. We have adapted fast candidate generation to work with our item set merging logic. We have achieved this in two complementary steps explained below.

First step. For any candidate item set that has at least 2 items in either part (V_s or V_i), we can use ordinary fast candidate generation over the frequent item sets in level k that have items in both V_s and V_i sets. After that, we check for a candidate C if $C \cap V_s$ is frequent in the replicated database, which is the output of phase 2, and $C \cap V_i$ is frequent in independent database, which is the output of phase 3.

Second step. Consider a $(k+1)$-length candidate C with one item x in one part and k items in the other part. Not all of its k-length supports have at least one item in either part, therefore, C cannot always be generated from the frequent item sets between parts in level k with fast candidate generation. We make use of the observation that if C is frequent, V_s will have k k-length subsets that include x. While traversing the $(k-1)$ length prefixes in the prefix tree, for each item x, we construct a set of conditional $(k-1)$-length patterns that have items in only one part by removing x. Then, for each item x, we generate k-length candidates from the corresponding set of conditional patterns using fast candidate generation. These k-length candidates have items in only one part and are checked if they are already frequent in that part. If so, then we add x back to generate C and apply the Downward Closure pruning restricted to k-length subsets across both parts.

After fast candidate generation, we use the ordinary caching and intersection routines of *Bitdrill* to calculate frequencies. Iteration continues until exhaustion of merged patterns. Note that this step can be used for any distribution of items, not just for our GPVS-based distribution.

3.4 Repl-Bitdrill Algorithm

The phase of mining replicated items in parallel can be considered as a stand-alone parallel FIM algorithm, which is similar to the second phase of *ParDCI* [10]. When used on its own, we call it *Repl-Bitdrill* as it replicates the tidlists of *all* frequent item sets on *all* processors, at the level that it starts mining. Note that *NoClique2* degenerates to *Repl-Bitdrill* when partitioning is impossible, i.e., all items are replicated. *Repl-Bitdrill* is used in Section 4 to experimentally show the merits of partitioning in *NoClique2*.

3.5 Comparison with Par-Eclat

To put things in perspective, it may help to note the ancestry of our algorithm. Our algorithm is close to *Par-Eclat* [5]. The most important similarities between two algorithms are: 1) We use the same graph of two items when $l=2$. 2) We also use graph theoretic observations to cluster items. 3) We also distribute items so that each processor mines independently with no further communication. On the other hand, we highlight the following differences:

TABLE 1
Speedup Values

Database	4 processors			8 processors		
	NC2	RBD	PDCI	NC2	RBD	PDCI
T60.I10.2000K	3.4	2.3	1.2	4.3	4.0	2.3
user-likesmovies	2.9	2.9	2.8	5.0	5.0	7.4
trec	2.6	2.7	–	4.2	4.5	–
trec.lp.200000	2.8	2.8	0.7	4.9	4.9	1.5

Database	16 processors			32 processors		
	NC2	RBD	PDCI	NC2	RBD	PDCI
T60.I10.2000K	7.7	6.2	4.4	11.1	2.6	8.7
user-likesmovies	7.5	7.9	11.4	10.7	10.6	8.2
trec	6.3	5.1	–	8.7	7.1	–
trec.lp.200000	7.9	1.6	2.8	13.0	1.7	5.2

1. We propose a novel item set clustering method based on GPVS.
2. Our item distribution method can minimize data replication by setting vertex weights appropriately.
3. Our algorithm is fairly independent of the underlying serial mining algorithm (but needs further work to implement phases 2 and 4 of *NoClique2*).
4. Work over replicated items is parallelized.

3.6 Implementation

Our implementations of *Bitdrill*, *Repl-Bitdrill*, *NoClique*, and *NoClique2* are written in C++ using MPI. The computation of GPVS in *NoClique2* is relevant to the experiments. We use the hypergraph partitioning-based formulation for computing a GPVS of G_{F_l} [23], [24]. To that end, we use the hypergraph partitioner *PaToH* [25], [26].

3.7 Applicability to Dense Data

We have indicated that our algorithm is not supposed to work well with problem instances that give rise to a dense graph. In dense databases, this is not necessarily the case, and we have observed that our method works even with such databases. When the graph is quite dense, a large number of items are replicated, and our algorithm degenerates into an algorithm like the second phase of *kDCI* that replicates all items. Often, choosing a more suitable support threshold or a starting level for our algorithm helps.

4 Experiments

Here, we give a summary of our experiments; more detailed experimental results are given in Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.32. We have run our algorithms *NoClique2* and *Repl-Bitdrill* as well as *ParDCI* on one synthetic (T60.I10.2000K) and three real-world databases on a Beowulf cluster. In Table 1, *NoClique2*, *Repl-Bitdrill*, and *ParDCI* are abbreviated as NC2, RBD, and PDCI, respectively. As seen in Table 1, out of 16 parallel mining cases, *NoClique2* achieves considerably higher speedup in eight cases, whereas *NoClique2* and *Repl-Bitdrill* attain close speedups in eight cases. *ParDCI* achieves the highest speedup in two cases. For the trec database, *ParDCI* unfortunately crashed, and we could not measure its running time. We would expect it to have good performance as in user-likesmovies which is similarly dense. For the sparser database T60.I10.2000K, *NoClique2* achieves

better speedups than the other algorithms. Only *NoClique2* attains increasing speedup with increasing number of processors for all the databases. *Repl-Bitdrill* and *ParDCI* show this nice property only for two databases each.

Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.32, contains a detailed explanation of the databases, experimental setup, speedup, partitioning quality, running time dissection, speedups of *NoClique* parallelizations, and discussion of observed superlinear speedups in *NoClique*. With regards to partitioning quality, we have examined expected versus actual load imbalance and data replication ratio. We have seen that our heuristic load estimates work but could be much improved. Data replication is controlled well enough but it is better for small number of processors. It turns out that in the sparse database, independent mining phase dominates and in the dense databases (user-likesmovies and trec) the collective work phase dominates. For these databases, the replication approach of *Repl-Bitdrill* and *ParDCI* is effective. However, in an important other case (trec.lp.200000) which represents the "long tail" in a real-world data set, there is a mixture of both phases, and ultimately *NoClique* does much better than *Repl-Bitdrill* and *ParDCI*, showing the true potential of our approach. The trec.lp.200000 database contains items in the trec with a frequency of 200,000 and lower. In the trec database, it is not possible to mine frequent item sets beyond a narrow set of items due to the power-law-like distribution of items, however in such real-world databases we are interested in relationships among a large number of items.

5 Conclusions

We have proposed an item distribution method that depends on theoretical observations that identify lack of cliques among two sets of items in G_{F_2}. The mining problem is decomposed into independent subproblems using a GPVS model which encapsulates the minimization of task or data redundancy as well as computational load or storage balance. We showed that this model can be extended to n-way distribution and any level of mining. Based on our distribution model, we designed and implemented two parallel FIM algorithms called *NoClique* and *NoClique2*. Experiments with synthetic and real-world databases on a Beowulf cluster showed considerable speedups, thus affirming the validity of our model.

A discussion of future work is present in Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS. 2011.32.

Acknowledgments

This work is partially supported by The Scientific and Technological Research Council of Turkey under grant EEEAG-109E019 and COST Action IC080 ComplexHPC. Part of the experimental tests were carried out at the TUBITAK ULAKBIM High Performance Computing Center. The authors thank Claudio Luchesse for making *ParDCI* available to them. The authors thank Bart Goethals for providing the benchmark results of FIMI 2004 experiments. Thanks to anonymous reviewers for recommending many improvements.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94)*, pp. 487-499, 1994.

[2] M.J. Zaki, "Generating Non-Redundant Association Rules," *Proc. Knowledge Discovery and Data Mining Conf.*, pp. 34-43, 2000.

[3] D.-I. Lin and Z.M. Kedem, "Pincer Search: A New Algorithm for Discovering the Maximum Frequent Set," *Proc. Sixth Int'l Conf. Extending Database Technology*, pp. 105-119, 1998.

[4] R. Agrawal and J.C. Shafer, "Parallel Mining of Association Rules," *IEEE Trans. Knowledge and Data Eng.*, vol. 8, no. 6, pp. 962-969, Dec. 1996.

[5] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "Parallel Algorithms for Discovery of Association Rules," *Data Mining and Knowledge Discovery*, vol. 1, no. 4, pp. 343-373, 1997.

[6] D.W. Cheung, V.T. Ng, A.W. Fu, and Y.J. Fu, "Efficient Mining of Association Rules in Distributed Databases," *IEEE Trans. Knowledge And Data Eng.*, vol. 8, no. 6, pp. 911-922, Dec. 1996.

[7] O. Zaïane, M. El-Hajj, and P. Lu, "Fast Parallel Association Rule Mining without Candidacy Generation," *Proc. IEEE Int'l Conf. Data Mining (ICDM '01)*, Nov.-Dec. 2001.

[8] A. Rudra, R.P. Gopalan, and Y.G. Sucahyo, "Scalable Parallel Mining for Frequent Patterns from Dense Data Sets Using a Cluster of PCs," *Proc. Sixth Int'l Conf. Information Technology*, Dec. 2003.

[9] E.-H. Han, G. Karypis, and V. Kumar, "Scalable Parallel Data Mining for Association Rules," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 3, pp. 337-352, May 2000.

[10] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "An Efficient Parallel and Distributed Algorithm for Counting Frequent Sets," *Proc. Fifth Int'l Conf. High Performance Computing for Computational Science (VECPAR '02)*, 2003.

[11] C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 1, pp. 21-36, Jan. 2006.

[12] H. Li, Y. Wang, D. Zhang, M. Zhang, and E.Y. Chang, "Pfp: Parallel fp-Growth for Query Recommendation," *Proc. ACM Conf. Recommender Systems (RecSys)*, P. Pu, D.G. Bridge, B. Mobasher, and F. Ricci, eds., pp. 107-114, 2008.

[13] A. Savasere, E. Omiecinski, and S.B. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. 21st Int'l Conf. Very Large Data Bases (VLDB '95)*, pp. 432-444, Sept. 1995.

[14] J.W.H. Liu, "A Graph Partitioning Algorithm by Node Separators," *ACM Trans. Math. Software*, vol. 15, no. 3, pp. 198-219, Sept. 1989.

[15] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Jan. 1999.

[16] A. Grama, V. Kumar, A. Gupta, and G. Karypis, *An Introduction to Parallel Computing: Design and Analysis of Algorithms*, second ed., Addison-Wesley, 2003.

[17] R.J. Bayardo Jr., "Efficiently Mining Long Patterns from Databases," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 85-93, 1998.

[18] S. Orlando, C. Lucchese, P. Palmerini, R. Perego, and F. Silvestri, "kDCI: A Multi-Strategy Algorithm for Mining Frequent Sets," *Proc. Second IEEE Int'l Conf. Data Mining (ICDM) Workshop Frequent Itemset Mining Implementations (FIMI '04)*, 2004.

[19] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Adaptive and Resource-Aware Mining of Frequent Sets," *Proc. IEEE Int'l Conf. Data Mining (ICDM '02)*, pp. 338-345, Dec. 2002.

[20] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets," *Proc. Second IEEE Int'l Conf. Data Mining (ICDM) Workshop Frequent Itemset Mining Implementations (FIMI '04)*, 2004.

[21] A. Fiat and S. Shporer, "AIM2: Another Itemset Miner," *Proc. Second IEEE Int'l Conf. Data Mining (ICDM) Workshop Frequent Itemset Mining Implementations (FIMI '04)*, 2004.

[22] E. Özkural and C. Aykanat, "A Space Optimization for FP-Growth," *Proc. Second IEEE Int'l Conf. Data Mining (ICDM) Workshop Frequent Itemset Mining Implementations (FIMI '04)*, 2004.

[23] Ü.V. Çatalyürek and C. Aykanat, "Hypergraph-Partitioning-Based Sparse Matrix Ordering," *Proc. Second Int'l Workshop Combinatorial Scientific Computing (CSC '05)*, June 2005.

[24] U. Catalyurek, C. Aykanat, and E. Kayaaslan, "Hypergraph Partitioning-Based Fill-Reducing Ordering," Technical Report BU-CE-0904, Bilkent Univ. Inst. of Science and Eng., 2009.
[25] Ü.V. Çatalyürek and C. Aykanat, "PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0," technical report, Bilkent Univ., Computer Eng. Dept., 1999.
[26] Ü.V. Çatalyürek and C. Aykanat, "Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 7, pp. 673-693, July 1999.

Eray Özkural received the BSc and MSc degrees from the Computer Engineering Department of Bilkent University, Ankara, Turkey. He is currently working toward the PhD degree from Bilkent University. His research interests include parallel computing, data mining, information retrieval, algorithms, machine learning, algorithmic information theory, artificial general intelligence, and philosophy of mind. He has worked as a parallel computing researcher at the European Union SEE-GRID project and Erendiz Süperbilgisayar Ltd. Among his recent innovations is a memory system for artificial general intelligence.

Bora Uçar received the PhD degree in computer engineering from Bilkent University, Ankara, Turkey, in 2005. He worked as a postdoctoral researcher at the Mathematics and Computer Science Department of Emory University, Atlanta, and at the Parallel Algorithms Project, CERFACS, France. Since January 2009, he is conducting research activities at ENS Lyon, France, as a research scientist of CNRS. His research interests include combinatorial scientific computing, parallel and high-performance computing, and sparse matrix computations.

Cevdet Aykanat received the BS and MS degrees in electrical engineering from Middle East Technical University, Ankara, Turkey, and the PhD degree in electrical and computer engineering from Ohio State University, Columbus. He was a Fulbright scholar during his PhD studies. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Ankara, Turkey, where he is currently a professor and the associate provost. His research interests mainly include parallel computing, parallel scientific computing and its combinatorial aspects, parallel computer graphics applications, parallel data mining, graph and hypergraph-theoretic models for load balancing, high-performance information retrieval systems, parallel and distributed databases, and grid computing. He has (co)authored about 60 technical papers published in academic journals indexed in ISI and his publications received above 400 citations in ISI indexes. He is the recipient of the 1995 Young Investigator award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science award. He was appointed as a member of IFIP Working Group 10.3 (Concurrent System Technology) in April 2004, as a member of the EU-INTAS Council of Scientists in June 2005, and as an associate editor of the *IEEE Transactions of Parallel and Distributed Systems* in December 2008.