Exploring Decomposition for Solving Pattern Mining Problems

YOUCEF DJENOURI, Dept. of Mathematics and Cybernetics, SINTEF Digital, Oslo, Norway
JERRY CHUN-WEI LIN, Dept. of Computing, Mathematics, and Physics, HVL, Bergen, Norway
KJETIL NØRVÅG, Dept. of Computer Science, NTNU, Trondheim, Norway
HERI RAMAMPIARO, Dept. of Computer Science, NTNU, Trondheim, Norway
PHILIP S. YU, Dept. of Computer Science, University of Illinois, Chicago, IL, United States.

Abstract This paper introduces a highly efficient pattern mining technique called Clustering-Based Pattern Mining (CBPM). This technique discovers relevant patterns by studying the correlation between transactions in the transaction database based on clustering techniques. The set of transactions is first clustered, such that highly correlated transactions are grouped together. Next, we derive the relevant patterns by applying a pattern mining algorithm to each cluster. We present two different pattern mining algorithms, one applying an approximation-based strategy and another based on an exact strategy. The approximation-based strategy takes into account only the clusters, whereas the exact strategy takes into account both clusters and shared items between clusters. To boost the performance of the CBPM, a GPU-based implementation is investigated. In order to evaluate the CBPM framework, we perform extensive experiments on several pattern mining problems. The results from the experimental evaluation show that the CBPM provides a reduction in both the runtime and memory usage. Also, CBPM based on the approximate strategy provides good accuracy, demonstrating its effectiveness and feasibility. Our GPU implementation achieves significant speedup of up to 332x on a single GPU.

Additional Key Words and Phrases: Pattern Mining, Decomposition, Scalability, GPU

ACM Reference Format:
Youcef Djenouri, Jerry Chun-Wei Lin, Kjetil Nørvåg, Heri Ramampiaro, and Philip S. Yu. 2019. Exploring Decomposition for Solving Pattern Mining Problems. *ACM Trans. Manag. Inform. Syst.* 1, 1 (January 2019), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Pattern Mining (PM) is a data mining technique that finds highly co-occurring items in a database in order to provide relevant patterns. Currently, various pattern mining techniques have been proposed, including Frequent Itemset Mining (FIM), Weighted Itemset Mining (WIM), Uncertain Itemset Mining (UIM), High Utility Itemset Mining (HUIM), and Sequential Pattern Mining (SPM). Frequent Pattern Mining (FPM) has largely been applied as a pre-processing step in several practical problem solving applications, such as market basket analysis [?], where FIM finds the correlation among products bought by different customers; information retrieval [?], where WIM and UIM

Authors' addresses: Youcef Djenouri, Dept. of Mathematics and Cybernetics, SINTEF Digital, Oslo, Norway; Jerry Chun-Wei Lin, Dept. of Computing, Mathematics, and Physics, HVL, Bergen, Norway; Kjetil Nørvåg, Dept. of Computer Science, NTNU, Trondheim, Norway; Heri Ramampiaro, Dept. of Computer Science, NTNU, Trondheim, Norway; Philip S. Yu, Dept. of Computer Science, University of Illinois, Chicago, IL, United States. youcef.djenouri@sintef.no,jerrylin@ieee.org, noervaag@ntnu.no,heri@ntnu.no,psyu@uic.edu.

2158-656X/2019/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

mine the correlations among terms of documents; business intelligence [?], where HUIM discovers the process models in the log of events; and bioinformatics [?], where SPM extracts the knowledge from the biological sequence data. As an example, considering the information retrieval problem, the collection of documents is transformed into a transaction database, where each document is considered as a transaction, each term as an item, and the tf-idf value for each term [?] as the weight or the probability of a given item. In this context, mining techniques, such as WIM and UIM, allow to study different correlations between pairs of terms in a document. For instance, if the pattern (*Knowledge*, *Engineering*) is relevant, a high dependency exists between the terms *Knowledge* and *Engineering*. Hence, if a user is looking for documents related to *Knowledge*, it would be useful to also return documents related to *Engineering*. Unfortunately, pattern mining techniques for large databases, such as FIM and WIM, suffer from long processing time (runtime). They are inefficient when solving complex problems, such as UIM, HUIM, and SPM. To reduce the runtime of pattern mining, several optimization techniques have been proposed [? ?]. However, these optimization techniques are incapable of dealing with databases containing a huge number of items, where only few of the relevant patterns are displayed to the end user. The main reason these techniques are inefficient is because they consider the whole database in the mining process.

1.1 Motivating example

Consider the trajectories of five buses illustrated in Figure 1. Each trajectory is mapped to the road map network of the United States. Trajectory pattern mining algorithms [?] consider the whole trajectories as sequences and apply the sequential pattern mining algorithms such as FAST [?], and/or other algorithms to identify the most frequent points (states in this case) shared by all the trajectories in the set. This allows to provide good guidance to users or decision makers in applications such as hot spot and crime detection [?], snapshot detection [?], etc. Considering the trajectories in Figure 1, the trajectories represented by the dashed lines cover three states (Minnesota, South Dakota, and Colorado), and the trajectories represented by the solid lines cover four other states (Illinois, Iowa, Nebraska, and Colorado). In addition, trajectories of the dashed lines only cover one state with the trajectories of the solid lines (Colorado). At a first glance, it is judicious to process the trajectories of dashed lines separately to the trajectories of the solid lines. Existing trajectory clustering algorithms deal with this problem by dividing the whole trajectories into similar clusters [?]. In our work, we attempt to follow this methodology by proposing a general framework to split the database into similar clusters and reduce the processing cost of the existing pattern mining algorithms. Existing partitioning-based pattern mining approaches [?] consider naive partitions of the transaction database among the sites for distributed processing. These algorithms ignore the correlation between the different transactions. For instance, with these algorithms, the trajectories of Figure 1 may be handled on the same site, with eight different states (items in this context) as problem size. This generates $2^8 - 1$ potential solutions. However, it could process the trajectories of the dashed lines on the same site with only three items, and the trajectories of the solid lines with only four items as problem size. This only generates $2^3 - 1$ potential solutions for trajectories of dashed lines and $2^4 - 1$ potential solutions for trajectories of solid lines.

1.2 Contributions

In this paper, we propose a divide and conquer approach based on splitting the problem into several small sub-problems, but as independent as possible, and then study and explore the correlation between them. The first challenge is to make the sub-tasks independent, i.e., to create highly correlated clusters with little overlapped on transaction contents, i.e., common items. The second challenge is how to address the missing patterns due to the overlap on transaction contents across

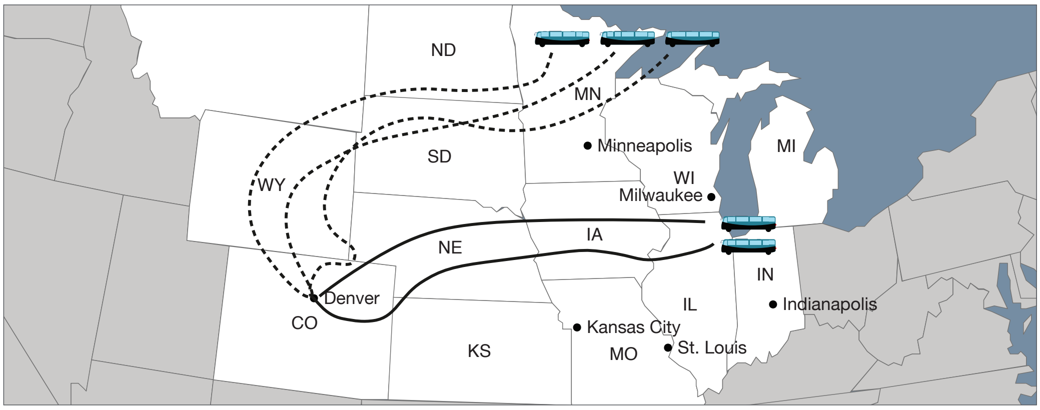


Fig. 1. Motivating example.

clusters. To deal with such challenging issues, we introduce a new framework called the clustering-based pattern mining (CBPM), which is a comprehensive extension of our previous work [?]. We developed two approaches, the approximate approach only addresses the first challenge, while the exact one addresses both. With this in mind, the main contributions of this work are as follows.

(1) We evaluate the use of different clustering algorithms to decompose the transaction database into highly correlated clusters, aiming at minimizing the number of the shared items between clusters: Naïve, HAC, k-means, bisecting k-means, and DBSCAN.
(2) We propose two novel strategies that use the clusters for pattern mining: an exact strategy that takes into account any shared items between clusters, and an approximate one that does not need to take into account the shared items.
(3) We investigate the impacts of applying both the exact and the approximate strategy on the mining effectiveness, as well as efficiency.
(4) We present a GPU-based implementation, and provide intelligent mapping between the GPU blocks and the clusters of transactions.
(5) We evaluate our approach by extensively studying the time complexity and comparing our approach with ten existing algorithms, applied on five different mining problems: FIM, WIM, UIM, HUIM, and SPM. This evaluation shows that our approach advances the state-of-the-art in terms of runtime, memory performance, as well as effectiveness. Moreover, our GPU implementation achieves significant speedup of up to 332x on a single GPU.

1.3 Outline

The remainder of the paper is organized as follows. Section 2 depicts the principles of pattern mining. Section 3 gives an overview of related work on the most important FPM variants. Section 4 provides a detailed explanation of our CBPM framework. Section 5 describes the GPU implementation of the CBPM framework. Section 6 presents the performance evaluation. Section 7 discusses the main findings from the application of the decomposition techniques to the pattern mining problems, and draws some future perspectives of using the proposed framework. Finally, Section 8 concludes the paper, and outlines the future work.

2 PRINCIPLES OF PATTERN MINING

In this section, we first present a general formulation of pattern mining, and then present a few pattern mining problems according to the general formulation.

Definition 2.1 (Pattern). *Let us consider $I = \{1, 2...n\}$ as a set of items, and $T = \{t_1, t_2...t_m\}$ as a set of transactions, where n is the number of items and m is the number of transactions. We define the function σ, where for the item i in the transaction t_j, the corresponding pattern reads* p=$\sigma(i, j)$.

Definition 2.2 (Pattern Mining). *Let us consider $I = \{1, 2...n\}$ as a set of n items, and $T = \{t_1, t_2...t_m\}$ as a set of m transactions.A pattern mining problem finds the set of all relevant patterns L, such as $L = \{p | Interestingness(T, I, p) \geq \gamma\}$. Note that the Interestingness (T, I, p) is the measure to evaluate a pattern p among the set of transactions T, and the set of items I, and where γ is the mining threshold.*

Any pattern mining problem could be written from the two previous definitions. For instance, i) Frequent Itemset Mining (FIM) [?] is defined by considering T as a Boolean database, and $Interestingness(T, I, p)=\frac{|p|_{T,I}}{|T|}$. ii) Weighted Itemset Mining (WIM) [?] is defined by considering T as a weighted database, and Interestingness(T, I, p)=$\sum_{j=1}^{|T|} W(t_j, I, p)$, where $W(t_j, I, p)$ is the minimum or the maximum weight of the items of the pattern p in the transaction t_j. iii) Uncertain Itemset Mining (UIM) [?] is defined by considering T as uncertain database, and Interestingness(T, I, p)=$\sum_{j=1}^{|T|} \prod_{i \in p} Prob_{ij}$, where $Prob_{ij}$ is the probability of the item i in the transaction t_j. iv) High Utility Itemset Mining (HUIM) [?] is defined by considering T as utility database, and $Interestingness(T, I, p) = \sum_{j=1}^{|T|} \sum_{i \in p} iu_{ij} \times eu(i)$. Note that iu_{ij} is the internal utility value of i in the transaction t_j, and $eu(i)$ is the external utility of each item i. v) Sequential Pattern Mining (SPM) [?] is defined by considering T as sequence database, and $Interestingness(T, I, p) = \frac{|p|_{T,I}}{|T|}$. Figure 2 shows an illustrative example of the pattern mining problems by considering the mining threshold as 50% for FIM, WIM, UIM, and SPM. For HUIM, we consider the mining threshold as 12, and the external utility values as $\{a : 2, b : 1, c : 3, d : 1\}$. For instance, if we assume the Apriori algorithm [?] on the FIM database, the process starts by generating the first candidate patterns of size 1, {a, b, c, d}. Then, the support of each candidate pattern is calculated. As an example, the support of the pattern a is equal to the number of occurrences of a over all numbers of transactions, which is equal to 60%. Its support is greater than the minimum support (50%), hence a is considered as frequent patterns. This process is repeated for all candidate patterns for size 1. The frequent patterns of this step is {a, b}. The next step aims to generate the candidate patterns of size 2 from the frequent patterns of size 1. The same process is repeated for all candidate patterns of size 2, this recursive process must be repeated until we get only an empty set of candidate patterns. The final result will be {a, b, ab}.

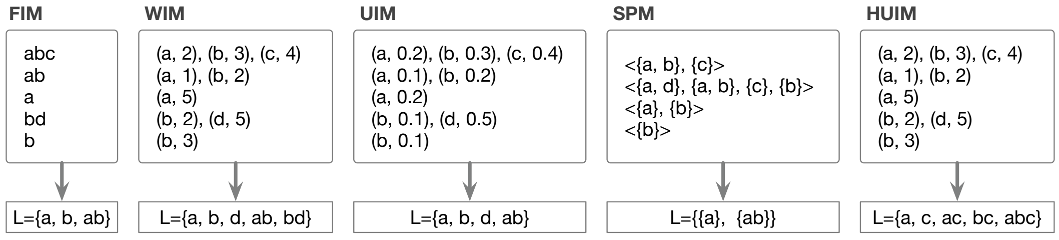


Fig. 2. Pattern mining problems.

3 RELATED WORK

This work is surrounded into two main topics, serial and parallel pattern mining algorithms, in the following, reviews on both topics are presented.

3.1 Serial pattern mining algorithms

Pattern mining problem has been largely studied over the past three decades [? ?]. Various pattern mining techniques have been reported, including the FIM, WIM, HUIM, UIM and SPM. The FIM is the first pattern mining problem which extracts all itemsets that exceed the minimum support threshold. Apriori [?] and FP-Growth[?] are the most used FIM algorithms. Apriori applies a *generate and test* strategy to explore the itemset space. The candidate itemsets are generated incrementally. To generate k-sized itemsets as candidates, the algorithm calculates and combines the frequent $(k-1)$-sized itemsets. This process is repeated until no candidate itemsets are obtained in an iteration. On the other hand, FP-Growth adopts a *divide and conquer* strategy, and compresses the transactional database into an efficient main-memory-based tree structure. It then applies recursively the mining process to find the frequent itemsets. The main limitation of the conventional FIM algorithms is the database format, where only binary cases could be mined. A typical application of this problem is the market basket analysis, where for a given transaction (customer), a given item (product) may be present or absent. To address this limitation, the WIM [?] was defined, where a weight is associated to each item to indicate its relative importance in the given transaction. The goal of WIM is to extract itemsets exceeding the minimum weight threshold. Yun [?] proposed weighted interesting pattern (WIP). It introduces an infinity measure that determines the correlation between the items of the same pattern.

The HUIM is an extension of the WIM where both internal and external utilities of the items are involved. The aim is to find all high utility patterns from the transaction database that exceeds the minimum utility threshold. The utility of a pattern is the sum of the utility of all its items, where the utility of an item is defined by the product by its internal and external utility values. Chan et al. [?] proposed the first HUIM algorithm. It applies the Apriori-based algorithm to discover top k high utility patterns. This algorithm suffers from the runtime performance, as the search space is not well pruned using the closure downward property. Thus, the utility measure is neither monotone nor anti-monotone. To address this limitation the transaction weighted utility (TWU) property is defined to prune the high utility pattern space [? ?]. It is an upper-bound monotone measure to reduce the search space. More efficient HUIM algorithms based on TWU have recently been proposed, such as EFficient high-utility Itemset Mining (EFIM) [?], and d^2HUP [?]. The particularity of such approaches is that they used more efficient data structures to determine the TWU and the utility values. The pattern mining has been applied to other applications, including UIM [?] and SPM [?]. UIM explores uncertain transaction databases, in which two models (expected-support and probabilistic itemsets) have been defined to mine uncertain patterns. Li et al. [?] proposed the probabilistic frequent itemset mining over streams. It derives the probabilistic frequent itemsets in an incremental way by determining the upper and the lower bounds of the mining threshold. SPM discovers a set of ordered patterns in a sequence database. Salvemini et al. [?] proposed the FAST algorithm. It finds the complete set of the sequence patterns by reducing the candidates generation runtime and employing an efficient lexicographic tree structure. Van et al. [?] introduced the pattern-growth algorithm in solving the sequential pattern mining problem with itemset constraints. It proposed an incremental strategy to prune the enumeration search tree, allows to reduce the number of visited nodes.

3.2 Parallel pattern mining algorithms

Regarding high-performance computing, many algorithms have been developed for boosting the FIM runtime performance [? ? ? ? ? ? ? ? ?]. However, few algorithms have been proposed for the other pattern mining problem [? ? ? ? ? ?]. In [?], GPApriori is developed by designed a "static bitset" memory structure to represent the transaction database on GPU architecture. In [?], CU-Apriori is proposed, which develops two strategies for parallelizing both candidate itemsets generation and support counting on a GPU. In the candidate generation, each thread is assigned two frequent $(k-1)$-sized itemsets, it compares them to make sure that they share the common $(k-2)$ prefix and then generates a k-sized candidate itemset. In the evaluation, each thread is assigned one candidate itemset and counts its support by scanning the transactions simultaneously. The evaluation of frequent itemsets is improved in [?] by proposing mapping and sum reduction techniques to merge all counts of the given itemsets. It is also improved in [?] by developing three strategies for minimizing the impact of the GPU thread divergence. In [?], a multilevel layer data structure is proposed to enhance the support counting of the frequent itemsets. It divides vertical data into several layers, where each layer is an index table of the next layer. This strategy can completely represent the original vertical structure. In a vertical structure, each item corresponds to a fixed-length binary vector. However, in this strategy, the length of each vector varies, which depends on the number of transactions included in the corresponding item. Several approaches have been proposed for solving the pattern mining problems using the MapReduce framework. In [?], the BigFIM algorithm is presented, which combines principles from both Apriori and Eclat. BigFIM is implemented using the MapReduce paradigm. The mappers are determined using Eclat algorithm, whereas, the reducers are computed using the Apriori algorithm. ?] apply the MapReduce framework for mining frequent biological sub-graphs. It first constructs the size-k subgraphs from the size-(k-1) subgraphs by the mappers, while the reducers will check whether or not the candidate subgraph meets the user-defined support. ?] present a parallel randomized algorithm for approximate pattern mining in the MapReduce framework. It starts by creating random samples from the whole set of transactions. Each mapper is assigned to one sample to generate the potential candidate patterns. The reducers then perform an aggregation function to determine the set of all approximate relevant patterns, which highly depend to the random samples created in the first stage. However, the authors only provide analytical guarantees regarding the quality of the approximate relevant patterns derived by this algorithm. ?] proposed a tree-based approach for mining uncertain data. It integrates the folk join framework by splitting the computationally intensive tasks into multiples pieces which can be solved in parallel. It also use a sampling method to transform the tree structure in a more compact one. This approach only finds a small number of relevant patterns due to the sampling process. ?] applied sequential pattern mining on large time series data, using the MapReduce framework. The time series data is transformed into several segments using statistical properties, such as mean and variance. Each segment is assigned to one mapper, to generate the suffix trees, and then extract the final times series patterns by the reducers. In [?], a Hadoop implementation based on MapReduce programming (FiDoop) was proposed for frequent itemset mining problem. It incorporates the concept of FIU-tree rather than the traditional FP-tree of used in the FP-Growth algorithm, for the purpose of improving the storage of the candidate itemsets. An improved version called FiDoop-DP was proposed in [?]. The authors proposed an efficient strategy to partition data sets among the mappers for minimizing data transfer cost between the different nodes. Voronoi diagram was used to minimize unnecessary redundant transactions transmission. kmeans was only used for selecting the Voronoi pivots. To the best of our knowledge, FiDoop-DP is the only work that explores data partitioning for performing pattern mining using the MapReduce, However, this approach uses partitioning during the map stage to re-organize the transactions among mappers for better exploration of cluster hardware architecture, and thus avoiding jobs redundancy. This task requires costly computational resources and it is not useful during the mining stage.

3.3 Discussion

The existing pattern mining algorithms consider the whole transaction databases to find the relevant patterns. They ignore the different dependencies and correlation between the transactions. Exploring the whole pattern mining problem require a huge time and memory consuming. In order to improve the performance of the pattern mining approaches, several techniques have been proposed, such as metaheuristics, which operate based on evolutionary and/or swarm intelligence approaches [?]. However, these techniques are incapable of dealing with large transaction databases, where only few interesting patterns may be discovered. To deal with this challenging issue, we will in this paper present a new framework for pattern mining algorithms. This new framework explores decomposition techniques for find out the relevant patterns. Similar ideas have been investigated in database community, in particular in the areas of record linkage and entity resolution [? ? ? ?]. The aim is to apply blocking-based techniques such as canopy clustering [?], suffix-blocking [? ?], and Q-gram based indexing [?], to derive the different records that represent the same real-world object in a given database, and check if such a real-world object may be determined by a single record. These methods need domain specific knowledge and require complete redesign for pattern mining applications. In addition, these approaches suffer from the accuracy problem, where the approximate heuristics are used on each block. In this paper, we attempt to follow these concepts by proposing a new framework for pattern mining problems, which can be used and guarantee the performance in terms of accuracy, memory and runtime. To boost the performance of our framework, a GPU-based approach is also investigated in this work.

4 CLUSTERING-BASED PATTERN MINING (CBPM)

This section presents the principle of the CBPM framework, and describes its components in details, separately. We finish this section by computing the theoretical complexity and showing an illustrative example of the CBPM framework.

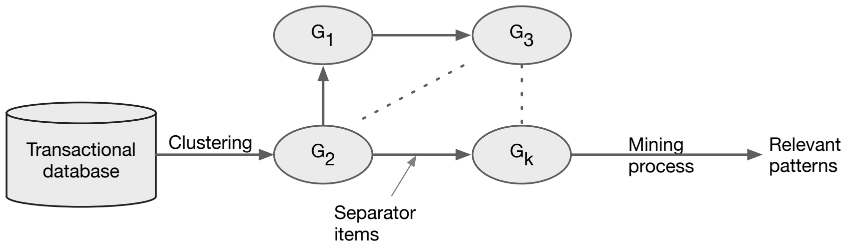


Fig. 3. The CBPM framework.

4.1 Overview

Here, we provide a general framework for the pattern mining for finding different dependencies between the transactions, which will be used for efficient improvement of the mining process. This framework illustrated in Figure 3 is composed of two main steps, i.e., the clustering and mining process, as follows.

(1) **Clustering.** In this step, a transaction database is divided into a set of homogeneous clusters using clustering techniques, where a cluster may be viewed as a subset of transactions of the whole set of transactions. We take advantage of the clustering technique to extract the relevant knowledge, which will be used by the pattern mining algorithms. The patterns shared by two clusters constitute a shared set. An interesting clustering approach is to minimize the size of the shared sets, while having in the same cluster transactions that are highly correlated, that is, transactions that share the maximum number of items. In this work, we will show different ways to decompose the transactions by investigating naïve, partitioning, hierarchical, density, and hybrid clustering. This allows to provide a clear picture of the decomposition step, and helps to make a fair conclusion about the most effective clustering algorithm for minimizing the number of shared items among the clusters of transactions.

(2) **Mining process.** The mining process is applied on the clusters found in the previous step. In this context, two main approaches have been investigated, i.e., the approximation-based and the exact approaches, i) In the approximate one the clusters are used to derive partial solutions, which are then merged into a global solution, and ii) In the exact approach, the mining process is applied on both the clusters and the shared sets, by aggregating these patterns on all clusters. It should be noted that, both approaches are applicable for all pattern mining algorithms.

4.2 Clustering

The set of transactions T is partitioned into k disjoint clusters $C = \{C_1, C_2 ... C_k\}$, where each cluster C_i is the subset of transactions in T such as $C_i \cap C_j = \emptyset$. Here, $I(C_i)$ is the set items of the cluster C_i and $I(C_i) = \{\bigcup I(t_j) / t_j \in C_i\}$

Proposition 4.1. *We define* C *as the set of clusters of the transaction database* T. *Suppose that the clusters in* C *do not share any items which means,* $\forall (i, j) \in [1...k]^2\ I(C_i) \cap I(C_j) = \emptyset$, *we have the following proposition:* $L = \{\bigcup_{i=1}^{k} L_i\}$. *Note that* L_i *is the set of the relevant patterns of the cluster* C_i.

Proof. Consider $\forall (i, j) \in [1...k]^2\ I(C_i) \cap I(C_j) = \emptyset$. *We have*
$\forall i \in [1...k]$: $L_i = \{p | Interestingness(T, I, p) \geq \gamma\}$. *The interestingness of the pattern p is based on its existence/absence in the whole transactions, so we have to compare p with all transactions in T, and return the transactions containing p for further processing. Now, consider a pattern p exists in* $I(C_i)$, *i,e* $p \subseteq I(C_i) \Rightarrow \forall e \in p, e \in I(C_i) \Rightarrow \forall e \in p, e \notin I(C_j), (\forall j \in [1...k], \vee j \neq i) \Rightarrow p \nsubseteq I(C_j) \Rightarrow$ $L_i = \{p | Interestingness(C_i, I(C_i), p) \geq \gamma\} \Rightarrow L = \{\bigcup_{i=1}^{k} L_i\}$ □

From the above proposition, one may argue that if the whole transactions are decomposed in such a way, the independent clusters will be derived. It means that, any cluster of transactions share items with any other cluster, and therefore, the clusters could be mined separately. Unfortunately, such case is difficult to realize, as many dependencies may be observed between transactions. The aim of clustering transactions is to minimize the shared items between the clusters, where these shared items are called *Shared Items*. In this section, we adopt different clustering algorithms [? ? ? ?] in order to minimize the number of *Shared Items*. Before this, we propose the following concepts,

(1) Similarity computation. The similarity measure between two transactions t_i and t_j is computed as $D(t_i, t_j) = \max(|I(t_i)|, |I(t_j)|) - (|I(t_i) \cap I(t_j)|)$. Note that $I(t_i)$ denotes the set of items of the transactions t_i.

(2) Centroids updating. Let us consider the set of transactions of the cluster $C_i = \{t_1^{(i)}, t_2^{(i)}, ..., t_{|C_i|}^{(i)}\}$. The aim is to find a gravity center of this set which is also a transaction. Inspired by the centroid formula developed in [?], we compute the centroid μ_i. The frequency of each item is calculated for all the transactions of the cluster C_i. The length of the transaction center

is denoted by l_i, and corresponds to the average number of items of all transactions in C_i as $l_i = \frac{\sum_{j=1}^{|C_i|} |I(t_j^{(i)})|}{|C_i|}$. Afterwards, the items of transactions in C_i are sorted according to their frequency, and only the l_i frequent items are assigned to μ_i, as $\mu_i = \{j | j \in \mathcal{F}_{l_i}\}$. Note that $\mathcal{F}_{l_i}$ denotes the set of the l_i frequent items of the cluster C_i.

(3) Transaction neighborhoods. We define the neighborhoods of a transaction t_i for a given threshold ϵ, noted $\mathcal{N}_{t_i}$ by $\mathcal{N}_{t_i} = \{t_j | D(t_i, t_j) \geq \epsilon \vee j \neq i\}$.

(4) Core transaction. A transaction t_i is called core transaction if there is at least the minimum number of transactions σ_T such as $|\mathcal{N}_{t_i}| \geq \sigma_T$.

(5) Shared items determination. After constructing the clusters of transactions, we have to determine the shared set of items between the clusters. We define the shared set of items, denoted by S, as $S = \bigcup_{i=1, j>i}^{k} I(C_i) \cap I(C_j)$. Moreover, we denote $S^{i,j}$ as the shared set between the clusters C_i and C_j.

4.2.1 Naive grouping for transaction decomposition. The naive grouping aims to group transactions into k disjoint clusters without processing. Given m transactions, $\{t_1, t_2 ... t_m\}$, the first $\frac{m}{k}$ transactions are assigned to C_1, the second $\frac{m}{k}$ transactions are assigned to C_2, and so until the last $\frac{m}{k}$ transactions are assigned to C_k.

4.2.2 Hierarchical agglomerative clustering for transaction decomposition. HAC (Hierarchical Agglomerative Clustering) [?] for transaction decomposition aims to create a tree-like nested structure partition $\mathcal{H} = \{\mathcal{H}_1, \mathcal{H}_2 ... \mathcal{H}_h\}$ of the data such that, $\forall (i,j) \in [1..k]^2, \forall (m,l) \in [1...h]^2, C_i \in \mathcal{H}_m, C_j \in \mathcal{H}_l, m \geq l \Rightarrow C_i \in C_j \wedge C_i \cap C_j = \emptyset$. It starts with all transactions in a separate cluster and then repeatedly joins the two clusters that are most similar until there is only one cluster. The similarity between two clusters C_i, and C_j is determined by the number of shared items between them, as $|I(C_i) \cap I(C_j)|$.

4.2.3 K-means for transaction decomposition. K-means [?] for transaction decomposition aims to optimize the following function: $J = \sum_{j=1}^{k} \sum_{t \in C_j} |t - \mu_j|^2$, where μ_j is the centroid of transactions in C_j. First, the transactions are assigned randomly to the k clusters and a centroid is computed for each cluster. Then, every transaction is assigned to a cluster whose centroid is the closest to that transaction. These two steps are repeated until there is no further assignment of the transactions to the clusters. In this work, we attempt to adapt the k-means algorithm for clustering of the transactional database.

4.2.4 Bisecting k-means for transaction decomposition. The bisecting k-means [?] for transaction decomposition uses a hybrid partitioning and divisive hierarchical approach. It starts with one cluster and at each step splits one of the clusters into two using the standard k-means algorithm. The process of bisecting a cluster is repeated several times, where the split that produces a higher similarity is selected.

4.2.5 DBSCAN for transaction decomposition. The DBSCAN algorithm [?] for transaction decomposition aims to search for clusters by checking the ϵ-neighborhood of each transaction. After the core transactions are determined, DBSCAN then iteratively collects density-reachable transactions from these core transactions directly, which may involve merging a few density-reachable clusters. The process terminates when no new transactions can be added to any cluster.

4.3 Mining Process

This step benefits from the knowledge extracted in the previous step. Instead of mining the whole transaction database with the full set of items. Each cluster of transactions with its items is handled separately. In this context, the two following strategies are proposed.

4.3.1 Approximation-based strategy. In this strategy, the clusters are handled separately without considering the shared items. The local relevant patterns are first extracted by applying the mining process on each cluster. The merging function is then used to derive the global relevant patterns. This function is constituted of the concatenation of all local relevant patterns. Such an approach returns partial relevant patterns from the whole transaction database. This is due to the fact that the shared items were not taken into account in the mining process. Algorithm 1 presents the pseudo-code of the approximation-based strategy.

Algorithm 1 Approximation-based strategy

```
1: Input:
   $C = \{C_1, C_2..., C_k\}$: The set of $k$ clusters
   $\gamma$: The mining threshold
2: Output:
   $\mathcal{A}$: The set of the relevant patterns discovered
3: $\mathcal{A} \leftarrow \emptyset$.
4: for i=1 to k do
5:     $\mathcal{A}_i \leftarrow MiningProcess(C_i, I(C_i), \gamma)$.
6:     $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}_i$
7: end for
8: return $\mathcal{A}$
```

Proposition 4.2. *An upper bound, respectively, the lower bound, of the number of the relevant patters discovered by the approximation-based strategy, are $|L|$, and $|L| - (\sum_{i=1}^{k} \sum_{j=(i+1)}^{k} (2^{|S_{ij}|} - 1))$, and we note $|L| - (\sum_{i=1}^{k} \sum_{j=(i+1)}^{k} (2^{|S_{ij}|} - 1)) \leq |\mathcal{A}| \leq |L|$*

Proof. In the worst case, the number of missing patterns of the approximation-based strategy is equal to the number of candidate patterns from the shared items between all clusters. This is may be realized, where the interestingness value of all the candidate patterns exceeds the mining threshold γ. In this case, the number of relevant patterns of the approximation-based strategy is equal to $|L|$ minus all the number of candidate patterns derived from the shared items of all clusters, equal to $\sum_{i=1}^{k} \sum_{j=(i+1)}^{k} (2^{|S_{ij}|} - 1)$. In the case of the candidate patterns derived from the shared items of all clusters are not relevant, the number of relevant patterns of the approximation-based strategy is $|L|$. □

From the above proposition, one may argue that the quality of the approximation-based strategy highly depends on the number of shared items of all clusters. If the number of the shared items is minimized, the approximation-based strategy is able to find all relevant patterns. This will be fixed by choosing well the number of clusters of the k-means algorithm or the ϵ value for DBSCAN algorithm.

4.3.2 Exact strategy. The goal of this strategy is to capture the missing patterns not covered by mining the local clusters. It considers the shared items as well as the clusters in the mining process. This allows to discover all relevant patterns from the whole transactions. The mining process is first applied on each cluster of transactions to extract the local relevant patterns. The possible candidate patterns are then generated from the shared items. For each generated pattern, the postprocessing function (see Def. 4.1) is then used to determine the interestingness of this pattern in the whole transaction database. Note that, the interestingness depends on the problem. For instance, if we are interested to deal with a frequent itemset mining problem, the interestingness function should be the support measure. The relevant patterns of the shared items are then concatenated with the local relevant patterns of the clusters to derive the global relevant patterns of the whole transaction database. Algorithm 2 presents the pseudo-code of the exact strategy.

Algorithm 2 Exact strategy

```
1: Input:
   $C = \{C_1, C_2 \ldots, C_k\}$: The set of $k$ clusters
   $S$: The set of shared items
   $\gamma$: The mining threshold
2: Output:
   $L$: The set of all relevant patterns
3: L ← ∅.
4: for i=1 to k do
5:     $L_i \leftarrow MiningProcess(C_i, I(C_i), \gamma)$.
6:     $L \leftarrow L \cup L_i$
7: end for
8: P ← ∅.
9: for each $S^{i,j} \in$ S do
10:     $P \leftarrow P \cup GenerateAllPatterns(S^{i,j})$.
11: end for
12: for each p ∈ P do
13:     if $\mathcal{F}(p) \geq \gamma$ then
14:         $L \leftarrow L \cup \{p\}$
15:     end if
16: end for
17: return $L$
```

Definition 4.1. *We define an postprocessing function of the pattern* p *in the clusters of the transactions* C *by* $\mathcal{F}(p) = \sum_{i=1}^{k} Interestingness(C_i, I(C_i), p)$

4.4 Complexity

The time complexity of the CBPM framework depends on the clustering and the pattern mining algorithms used in the overall process. We assume that the complexity of the k-means algorithm [?], or the complexity of DBSCAN algorithm [?] requires $O(m \times n)$. Considering the mining process, We define the complexity of any pattern mining algorithm A by $O(\text{Cost}(A, n, m))$. Note that m and n are the number of transactions and the number of items, respectively. Two possible cases are as follows.

4.4.1 Approximation-based strategy. In this strategy, the mining is applied on each cluster without considering the shared items. The complexity of the CBPM using this strategy is $O((n \times m) + \sum_{i=1}^{k} Cost(A, |C_i|, |I(C_i)|))$

4.4.2 Exact strategy. In this strategy, the mining is applied on each cluster where the shared items are taken into account. The cost of constructing the shared items reads $O(k^2)$. Here, the postprocessing function is performed for each shared itemset, so that the complexity of this function is $O(k \times |S|)$, where S is the set of the shared items. Thus, the complexity of the CBPM using this strategy is
$O((n \times m) + (k^2) + (\sum_{i=1}^{k} Cost(A, |C_i|, |I(C_i)|)) + (k \times |S|))$

Table 1 compares the complexity of some of the existing pattern mining algorithms using the CBPM framework by varying the function $Cost(A, m, n)$. Note that, the worst complexity is computed by considering the maximum number of transactions, the average number of transactions, and the size of the shared items as n. For simplicity, we assume the same number of transactions and items on each cluster (i,e $\forall i \in [1 \ldots k], |C_i| = m/k \wedge |I(C_i) = n/k$). From this table, we may conclude

Table 1. Complexity of the existing pattern mining algorithms using the CBPM framework.

Problem	Algorithm	Cost(A, m, n)	CBPM (Exact)	CBPM (Approximate)
FIM	Apriori[?]	mn^2	$\frac{mn^2+nk+k^4}{k^2}$	$\frac{mn^2}{k^2}$
	FP-Growth[?]	n^2	$nm + kn^2 + k^2$	$nm + kn^2$
	PrePost+[?]	$nlog(n)$	$\frac{nm+nlog(n)}{k} + k^2$	$\frac{nm+nlog(n)}{k}$
	SSFIM[?]	$m2^n$	$m2^{n/k} + nk + k^2$	$n2^{n/k}$
WIM	**WFIM**[?]	$mnlog(n)$	$\frac{mnlog(n/k)}{k} + nk + k^2$	$\frac{mnlog(n/k)}{k}$
	WIP[?]	mn^2	$\frac{mn^2+nk+k^4}{k^2}$	$\frac{mn^2}{k^2}$
UIM	**U-Apriori**[?]	$mn^2log(n)$	$\frac{mn^2log(n/k)}{k} + nk + k^2$	$\frac{mn^2log(n/k)}{k}$
HUIM	d^2HUP[?]	$n^4log(n)$	$\frac{n^4}{k^3}log(n/k) + nm + nk + k^2$	$\frac{n^4}{k^3}log(n/k) + nm$
	EFIM[?]	$n^3 + log(n^2)$	$\frac{n^3}{k^2} + \frac{log(n^2/k^2)}{k} + nm$	$\frac{n^3}{k^2} + \frac{log(n^2/k^2)}{k} + nm + nk + k^2$
SPM	**FAST**[?]	n^4	$\frac{n^4}{k^3} + nm + nk + k^2$	$\frac{n^4}{k^3} + nm$

that by using the CBPM framework, the complexity of all algorithms is reduced k orders of the magnitude. In addition, the Pre-Post+, WFIM, U-Apriori, EFIM, and FAST are the best algorithms of the pattern mining problems (FIM, WIM, UIM, HUIM, and SPM). Thus, these algorithms are considered as baselines in the experimentation section.

4.5 Example

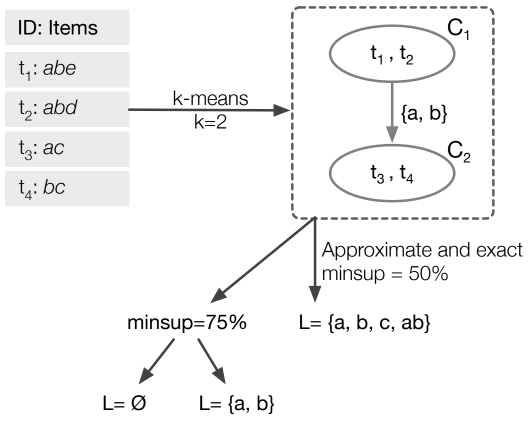


Fig. 4. Illustrative example.

Figure 4 presents an illustrative example of the CBPM framework for solving the frequent itemset mining problem. In this case, a pattern is viewed as an itemset (set of items), with a Boolean value (present or absent) in the given transaction. The transaction database is first partitioned using any clustering algorithm, without loss of generality, in this example, we used the k-means algorithm (with $k = 2$). Two clusters are found, i.e., $C_1 = \{t_1, t_2\}$ with $I(C_1) = \{a, b, d, e\}$, and $C_2 = \{t_3, t_4\}$ with $I(C_2) = \{a, b, c\}$. The shared items between C_1 and C_2 are $\{a, b\}$. For instance, if the minimum support is set to 50%, the approximate and the exact strategies return the same result $L = \{a, b, c, ab\}$: i) In the approximate strategy, the mining process is applied on the C_1 and C_2, we find $L_1 = \{a, b, ab\}$, and $L_2 = \{a, b, c\}$, the concatenation will be $L = \{a, b, c, ab\}$. ii) In the exact strategy, the same process is applied for C_1 and C_2, followed by the generation of all possible itemsets from the shared items, which results in $L_1 \cup L_2 \cup \{a, b, ab\}$=$\{a, b, c, ab\}$. Now, if we consider the minimum support set to 75%, different results are found for the approximate and the exact strategies as follows. i) The approximate strategy could not find any frequent itemsets since each cluster contains only two transactions, whereas the minimum support is 75%. That means we have to find itemsets that appear at least three times, and the result will be empty set. ii) This issue will be solved by the exact strategy, where the shared items are explored. The possible candidate itemsets from the shared set is $\{a, b, ab\}$, the support of each itemset in this set is the postprocessing of supports of the clusters C_1 and C_2. For example, $\mathcal{A}(\{a\}) = support(C_1, I(C_1), \{a\}) + support(C_2, I(C_2), \{a\}) = 2/4 + 1/4 = 3/4$. The same process is applied for all candidate itemsets, and the result will be $\{a, b, ab\}$, which is exactly the same result reported by the Apriori algorithm, if we consider the whole transaction database.

5 PARALLEL IMPLEMENTATION

In this section, we first propose a generic approach to implement CBPM on parallel architectures. A particular instantiation on GPU architecture of this generic approach is then presented.

5.1 Generic Approach to Parallelize CBPM

To run CBPM on any parallel architecture, the following sequential steps have to be performed:

(1) **Partition the database**: in this step, the transaction database is divided into partitions, whereby each partition contains a set of transactions. Any partitioning algorithm could be used here. In this work, we adopt the five decomposition algorithms (naive grouping, HAC, k-means, bisecting k-means, and DBSCAN). This step is performed in the CPU.
(2) **Computing and storing the local results**: in this step, each parallel node apply a serial pattern mining algorithm on each cluster, generate all relevant patterns from the cluster of transactions that is assigned to it and stores them in the set of all relevant patterns. The latter is built following the same logic of building the list of the relevant patterns in the serial implementation of CBPM. As for the serial implementation, we have two variants, i) a parallel approximation-based strategy that does not consider the shared items, and ii) a parallel exact strategy, which considers the shared items. Once the local relevant patterns are calculated, they will be send it to CPU for further processing.
(3) **Merging the local results**: the local relevant patterns are merged into a global one on the CPU side. This can be done using a simple concatenation as is the case of parallel approximation-based strategy, or an postprocessing as the case of parallel exact strategy.

The instantiation of the three steps defined above must be carefully designed to fit the hardware in use. In the remainder of this section, an instantiation of this generic approach is presented using GPU hardware.

5.2 GPU-CBPM

GPUs (Graphic Processing Units) are graphical cards initially developed for efficient generation of images intended for a display device, but their use as a powerful computing tool has gained popularity in many domains during the last decade [**? ? ?**]. The hardware is composed of two hosts, i) the CPU and, ii) the GPU hosts. The former contains processor(S) and main memory. The latter is a multi-threading system that consists of multiple computing *cores*, where each core executes a block of threads. Threads of a block in the same core communicate with one another using a shared memory, whereas the communication between blocks relies on a global memory. The CPU/GPU communication is made possible by hardware buses. In the following, the adaptation of CBPM for deployment on GPU architectures is denoted GPU-CBPM. In GPU-CBPM (see Figure 5), the transaction database is first partitioned on k clusters $\{C_1, C_2 ... C_k\}$ using the decomposition

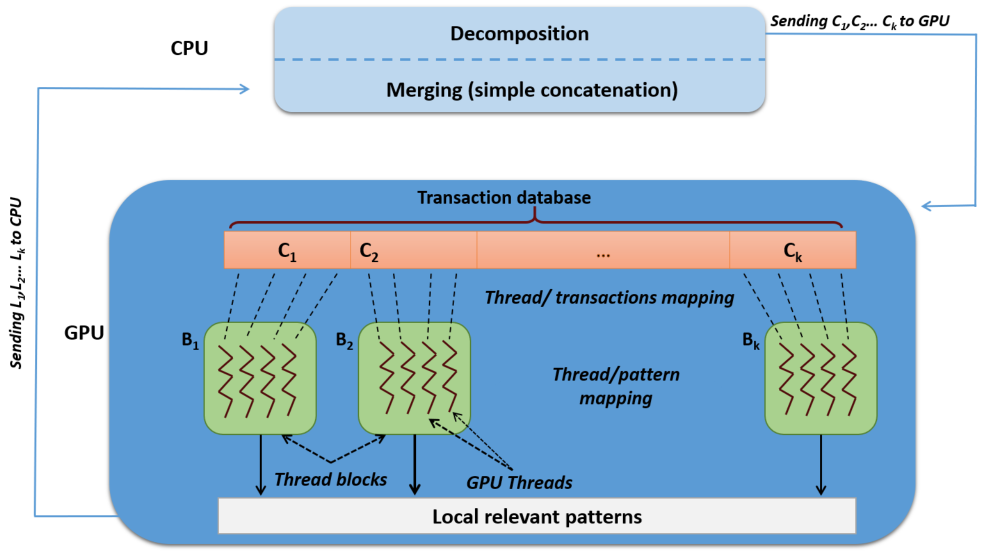


Fig. 5. GPU-CBPM framework.

methods. The set of designed clusters are then sent to GPU. Each block of threads is mapped onto one cluster, where the same mining process is applied on each block in parallel. If we consider the size of the shared memory of each block is r, the first r transactions of the cluster C_i are allocated to the shared memory of the block, and the remaining transactions of the cluster C_i is allocated to the global memory of the GPU host. GPU-CBPM defines a *local table*, $table_i$, for storing the relevant patterns of the cluster C_i. The local table of each cluster is sent to CPU for further processing. In this context, CPU host performs merging step to find the global relevant patterns. Two merging operators are defined. i) simple concatenation is applied for paralyzing the approximation-based strategy, it defines by the union of all sets of relevant patterns in the local tables, and ii) postprocessing is applied for paralyzing the exact strategy, it defines by applying the postprocessing function (see Def. 4.1) on the shared items S and the local tables. Algorithm 3 presents the pseudo-code of GPU-CBPM using standard CUDA operations.

From a theoretical standpoint, GPU-CBPM improves the serial implementation of CBPM by exploiting the massively threaded computing of GPUs while mining the clusters of transactions. GPU-CBPM also minimizes the CPU/GPU communication, by defining only two points of CPU/GPU communication. The first one takes place when the transaction database is loaded into the GPU host, and the second one when the local tables are returned to the CPU. GPU-CBPM also provides an efficient memory management by using different levels of memories including global and shared memories. However, GPU-CBPM may suffer from the synchronization between the GPU blocks. This takes place when the GPU blocks process clusters with different number of transactions. This issue degrades the performance of the GPU-based implementation of the CBPM framework. In real scenarios, different number of transactions per cluster may be obtained, this depends on the way of the clustering used in the decomposition step, as the size of the clusters are different, as the synchronization cost of the GPU-based implementation will be high. All these statements will be clearly explained in the performance evaluation section (See Section 6 for more details).

Algorithm 3 GPU-CBPM: CPU and GPU hosts

```
 1: /*************************CPU Host******************************************/
 2: Input:
    $T = \{t_1, t_2..., t_m\}$: The set of $m$ transactions
    $I = \{1, 2..., n\}$: The set of $n$ items
    $C = \{C_1, C_2..., C_k\}$: The set of $k$ clusters
    $S$: The set of shared items
    $\gamma$: The mining threshold
    $Algo$: The pattern mining algorithm
 3: Output:
    $L$: The set of all relevant patterns
 4: C ⟵ Decomposition(T, I)
 5: S ⟵ SharedItems(C)
 6: cudaMemcpy($C'$, C, n × m, cudaMemcpyHostToDevice) Mining«<k, 1024»>(L, $C'$, $\gamma$, Algo)
 7: if Approximation then
 8:     return L
 9: else
10:     P ⟵ ∅
11:     for each $S^{i,j}$ ∈ S do
12:         $P \leftarrow P \cup GenerateAllPatterns(S^{i,j})$.
13:     end for
14:     for each p ∈ P do
15:         if $\mathcal{F}(p) \geq \gamma$ then
16:             $L \leftarrow L \cup \{p\}$
17:         end if
18:     end for
19:     return L
20: end if
21: /*************************GPU Host******************************************/
22: Kernel Mining(L, $C'$, $\gamma$, Algo)
23: input
    $Shared$ T []: Array of transactions allocated in shared memories
24: Output:
    $L'$: The set of all relevant patterns of all blocks
25: idx ⟵ blockIdx.x × blockDim.x + threadIdx.x
26: T[idx] ⟵ $C'_{blockIdx.x}$[idx]
27: $L'$[blockIdx.x]=Algo(T, $\gamma$)
28: cudaMemcpy($L'$, L, $|L'|$, cudaMemcpyDeviceToHost)
```

6 PERFORMANCE EVALUATION

Intensive experiments have been carried out to evaluate the CBPM framework. First, the FIM, WIM, UIM, HUIM, and SPM problems have been investigated using standard datasets, by integrating the CBPM on the SPMF data mining library [?]. The CBPM java source code is integrated on the five best pattern mining algorithms in terms of the time complexity (See Sec. 4.4): i) Frequent itemset mining: PrePost+ [?], ii) Weighted itemset mining: WFIM[?], iii) Uncertain itemset mining: U-Apriori [?] , iv) High utility itemset mining: EFIM[?], and v) Sequential pattern mining: FAST[?]. Second, the results of CBPM framework on real taxi trajectory dataset has been shown and compared with the first phase of the RegMiner algorithm [?]. All serial implementations are done on a computer with 64 bit core i7 processor running Windows 10 and 16 GB of RAM. Finally, the GPU-based implementation is illustrated using sparse transaction databases.

6.1 Description of Standard Datasets

We perform the experiments using well-known pattern mining datasets[1]. Table 2 presents the characteristics of the standard datasets used in our experiments. Six datasets, i.e., Accident, Chess, Connect, Mushroom, Pumsb, and Korasak, Foodmart, and Chainstore, are used for evaluating the FIM algorithms. The first four datasets are used to evaluate the FIM algorithms. Further to the

[1]http://www.philippe-fournier-viger.com/spmf/

Table 2. Description of standard datasets.

Problem	Dataset Name	Trans.Size/ Sequence Count	Item Size	Aver. Size/ Avg. Seq. Size
FIM WIM HUIM	Accident	340,183	468	33.8
	Chess	3,196	75	37.0
	Connect	67,557	129	43.0
	Mushroom	8,124	119	23.0
	Pumsb	49,046	2,113	74.0
	Korasak	990,000	41,270	8.1
	Foodmart	4,141	1,559	4.4
	Chainstore	1,112,949	46,086	7.2
SPM	Leviathan	5,834	9,025	33.81
	Sign	730	267	51.99
	Snack	163	20	60
	FIFA	20,450	2,990	34.74

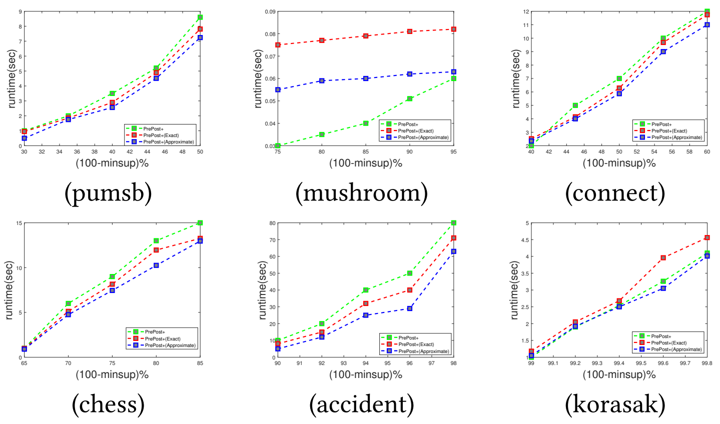


Fig. 6. Runtime of the PrePost+ with and without the CBPM framework.

FIM datasets, the last two datasets have been considered for evaluating the WIM, HUIM and UIM algorithms. For evaluating the three latter problems, we consider the following.

(1) WIM: Foodmart and Chainstore containing real weights. For FIM datasets, a generator function is used to generate the weights of the items as carried out in the previous work [**?**].
(2) UIM: The probabilities are generated using the normal distribution with a mean of 90% for high probability value, and 10% for low probability value with standard deviation of 5% for high probability value and 6% for the low probability value. This is done as in the previous work [**?**].
(3) HUIM: Foodmart and Chainstore are customer transaction databases containing the real external/internal utility values. For the FIM datasets, external/internal utility values have been respectively generated in the [1, 1,000] and [1, 5] intervals using a log normal distribution as done in the previous works [**? ?**].

The last four datasets, that is, Leviathan, Sign, Snake, and FIFA, are used to evaluate the SPM algorithms. In addition, an IBM Synthetic Data Generator for Itemsets and Sequences [2] is used to generate synthetic datasets of different number of items and transactions.

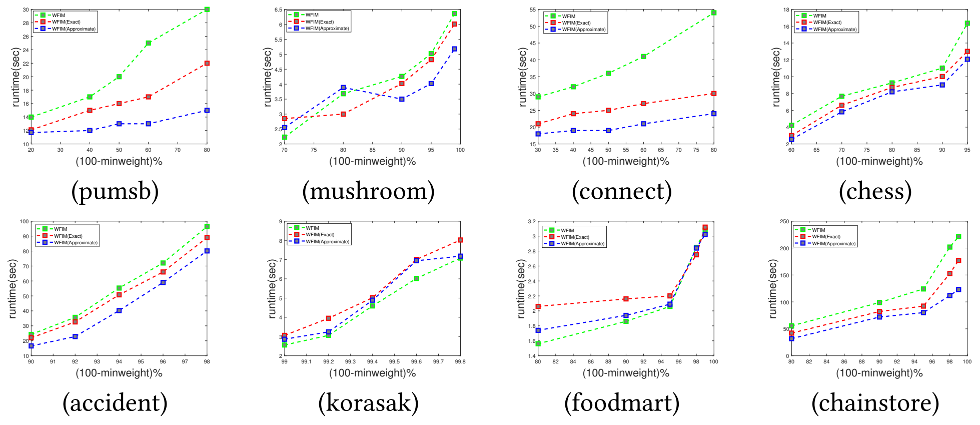


Fig. 7. Runtime of the WFIM with and without the CBPM framework.

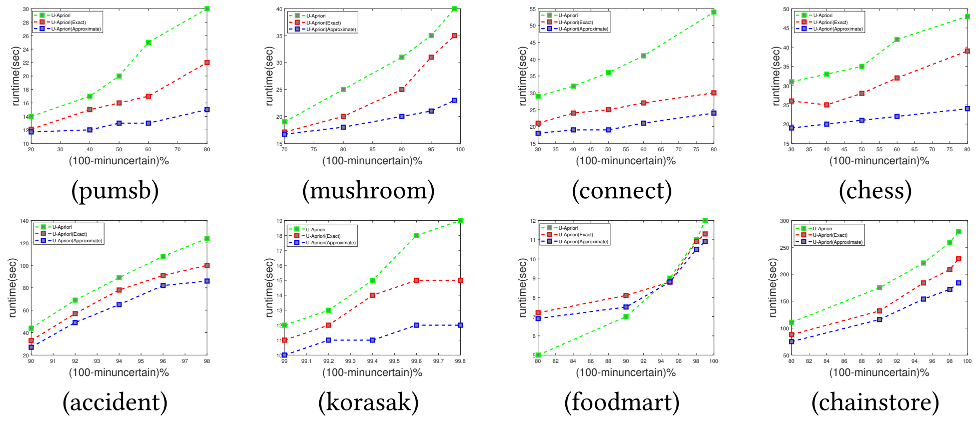


Fig. 8. Runtime of the U-Apriori with and without the CBPM framework.

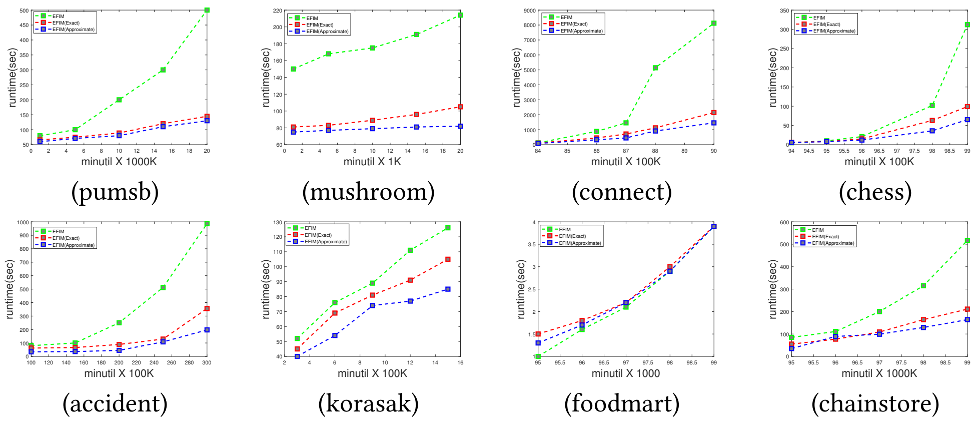
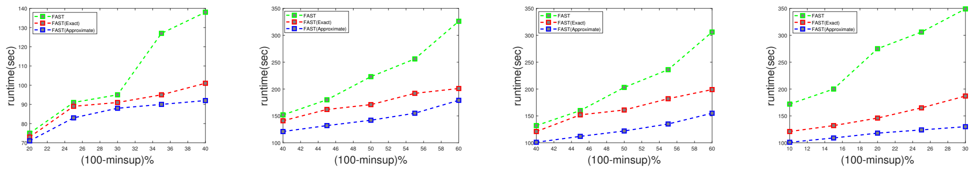


Fig. 9. Runtime of the EFIM with and without the CBPM framework.



(leviathan) (sign) (snack) (FIFA)

Fig. 10. Runtime of the FAST with and without the CBPM framework.

Table 3. Comparison of the maximum memory usage (MB), and the maximum number of visited nodes of the pattern mining algorithms with and without the CBPM framework.

		memory consumption			#visited nodes		
Problem: Algorithm	Dataset	Without CBPM	CBPM: Exact	CBPM: Approximate	Without CBPM	CBPM: Exact	CBPM: Approximate
FIM: PrePost+	pumsb	10	8	**7**	18,112	12,119	**5,127**
	mushroom	2	2	**2**	745,129	512,131	**598,748**
	connect	108	96	**64**	996,008	518,576	**296,748**
	chess	75	69	**51**	1,517,339	1,318,152	**800,563**
	accident	637	439	**332**	10,458	9,289	**6,780**
	korasak	143	99	**68**	1,851	1,847	**856**
WIM: WFIM	pumsb	732	661	**492**	35,845	31,785	**23,175**
	mushroom	51	42	**30**	1,874,457	1,798,214	**1,312,147**
	connect	308	278	**178**	1,524,333	912,127	**759,659**
	chess	179	152	**64**	2,685,417	2,000,110	**1,598,667**
	accident	879	821	**663**	26,556	22,996	**18,845**
	korasak	371	300	**253**	2,125	2,001	**1,002**
	foodmart	89	81	**47**	198,007	177,223	**135,168**
	chainstore	302	291	**200**	2,001	1,782	**1,096**
UIM: U-Apriori	pumsb	748	684	**527**	55,111	50,119	**42,219**
	mushroom	65	54	**41**	2,415,002	2,117,107	**1,658,127**
	connect	396	299	**201**	1,711,418	1,174,718	**817,147**
	chess	195	164	**88**	2,845,457	1,400,107	**1,000,748**
	accident	912	861	**719**	42,128	39,027	**35,187**
	korasak	401	328	**284**	2,517	2,314	**1,802**
	foodmart	101	91	**62**	221,127	197,117	**174,331**
	chainstore	419	379	**218**	2,927	2,241	**1,685**
HUIM: EFIM	pumsb	1075	912	**715**	59,597	51,578	**45,748**
	mushroom	112	106	**91**	3,179,165	2,743,258	**2,089,153**
	connect	567	478	**285**	1,952,111	1,112,553	**928,216**
	chess	218	153	**101**	3,334,258	2,957,514	**2,147,214**
	accident	1230	1112	**701**	55,211	40,128	**32,198**
	korasak	608	427	**217**	3,142	2,546	**2,336**
	foodmart	112	98	**75**	326,158	300,258	**257,845**
	chainstore	698	601	**411**	3,748	3,147	**2,415**
SPM: FAST	leviathan	245	211	**145**	5,298	4,958	**2,685**
	sign	375	351	**168**	6,510	5,882	**4,005**
	snack	417	412	**214**	8,222	7,984	**4,847**
	FIFA	749	695	**459**	10,214	9,002	**6,123**

6.2 Performance of the sequential version

Runtime. Figures 6, 7, 8, 9 and 10 present the runtime performance of the pattern mining algorithms with and without the CBPM framework for both approximate and exact strategies using different datasets and with different mining threshold. The results reveal that by reducing the mining threshold, and with increasing the complexity of the problem solved, the pattern mining algorithms benefit from the CBPM framework. Thus, for a low mining threshold, and for a more complex problem like UIM, HUIM or SPM, the approximation-based and exact strategies outperform the

[2]https://github.com/zakimjz/IBMGenerator

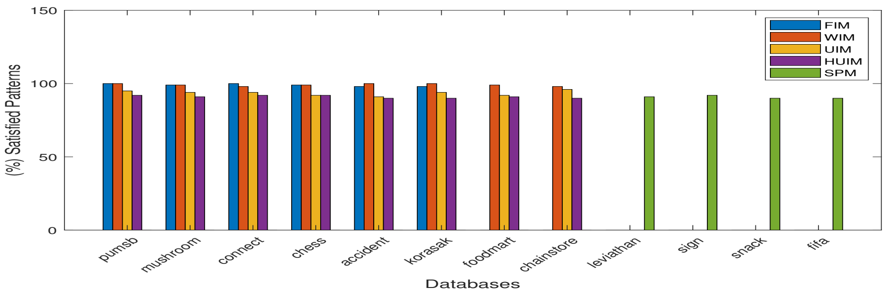


Fig. 11. Ratio of the satisfied patterns using the approximate strategy with the CBPM framework.

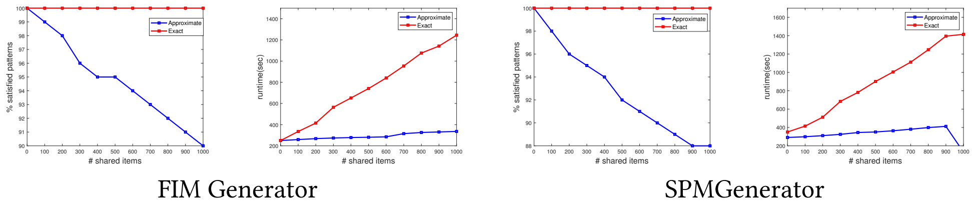


Fig. 12. Accuracy and runtime of approximate and exact-based strategies for different number of shared items.

original pattern mining algorithms. For instance, for the minimum utility threshold of $1600K$, the runtime of the original EFIM and EFIM using the CBPM framework is 1 second in the Connect dataset. However, by setting the minimum utility to $1000K$, the runtime of the original EFIM exceeds $8,000$ seconds, and the runtime of the EFIM with CBPM framework does not reach $1,500$ seconds. These results are achieved thanks to the following factors. i) The decomposition method applied to the CBPM framework by minimizing the number of the shared items. ii) Solving the sub-problems with small number of transactions and small number of items, instead of dealing the whole transaction database with the whole distinct items, and iii) The integrability of the pattern mining algorithms and the CBPM framework.

Memory consumption. In this experiment, the memory usage of the pattern mining algorithms with and without the CBPM framework is recorded. The results are measured using the Java API. Table 3 lists the maximum memory usage for a varying dataset used and the problem solved. From this table, we may observe that both exact, and approximate strategies outperform the previously reported pattern mining algorithms, for all datasets. Moreover, the pattern mining algorithms consume less of memory when using the CBPM framework. For instance, by running the EFIM algorithm on the chainstore dataset, the approximate strategy consumes 411 *MB*, while the original EFIM consumes 698 *MB* in average. The reason for efficient memory usage of the CBPM framework is because it deals only with small datasets at a time rather than other algorithms, while the conventional algorithms deal with the whole dataset. The CBPM explores small sub-trees, while conventional algorithms explore the whole tree for finding the relevant patterns. In addition to these results, the approximate strategy outperforms the exact strategy for all cases. This may be explained by the fact that the approximate strategy does not take into account the shared items in the search space, where a less memory is required for the overall mining process of such strategy.

Number of visited nodes. Another experiment has been carried out to investigate the pruning of the search space of the CBPM framework by comparing the maximum number of the visited nodes (patterns) of the search-enumeration tree by the pattern mining algorithms with and without the CBPM framework, and by exploiting both approximation-based and exact strategies. According to Table 3, the results reveal that by using the CBPM framework, the pattern mining algorithms efficiently prune the search space, while only sub-trees are explored against the whole tree for the original pattern mining algorithms. The results also show that the approximation-based strategy outperforms the exact one, for all cases. This is due to the fact that the approximate strategy ignores the shared items between the clusters, where the exact one generates all possible candidate patterns from the shared items.

Ratio of the satisfied patterns. This experiment evaluates the approximation-based strategy proposed in this work. Note that in the pattern mining literature, there are many approximation-based algorithms by exploiting the metaheuristics [**?**]. However, these approaches are out of the scope of this paper, where the main goal of this work is to show the effect of the decomposition on the pattern mining algorithms. Figure 11 presents the ratio of the satisfied patterns (i.e., patterns that exceed the mining threshold value). Note that, the last four databases are used for sequential pattern mining, and thus only one bar is obtained for these datasets. By varying the dataset used, and the pattern mining problem solved in the experiment, we show that the ratio of the satisfied patterns reach up to 90% for all cases. However, the ratio is different for each problem. Thus, there are problems, while the ratio of the satisfied patterns is up to 98% such as the FIM and WIM, whereas, there are other more complex problems, while the ratio of the satisfied patterns is between 98% and 90% such as the UIM, HUIM and SPM. These results are achieved thanks to the decomposition method employed in the CBPM framework by minimizing the number of the shared items, and the postprocessing function used in the approximation-based strategy. Figure 12 presents the ratio of the satisfied patterns, and the runtime using IBM Synthetic Data Generator for Itemsets and Sequences [3]. By varying with the number of shared items from 1 to $1,000$, the ratio of the satisfied patterns is reduced from 100% to 88% for the approximate based strategy, and the runtime of the exact strategy is increased from 200 to $1,500$. Thus, the number of shared items resulting from the decomposition method has a high impact of the accuracy of the approximate-based strategy, and also in terms of the runtime of the exact approaches. In fact, the approximate-based strategy only explores the clusters of transactions and ignores the shared items. Hence, it might be some relevant patterns in the set of shared items among the clusters. However, as the exact strategy considers both the clusters of transactions and the shared items among the clusters, this may increase the processing time compared to the approximate-based strategy. We can conclude that there is a trade-off between quality and runtime of our framework depending on the number of shared items. In general, if there is a high correlation among different transactions of a given database, the decomposition method may derive considerable number of shared items between different clusters. For this, we can say that if the ratio between the similarity of the transactions in the given database, and the similarity between the different transactions within the cluster is high, then our framework may fail, and give a bad result. Otherwise, our framework returns good results in terms of both runtime and accuracy.

Sensitivity to number of clusters. The aim of this experiment is to show the sensitivity of the number of clusters on the CBPM framework. In order to do this study, we explored the k-means algorithm on the FIM problem. We varied the number of clusters from 1 to 25 on the FIM transaction databases, and we computed the accuracy and the runtime for the approximate strategy (after 25 clusters, no changes in accuracy is observed). Figures 13 show the runtime and the accuracy, computed by the percentage of the satisfied patterns, of the CBPM framework using the approximation strategy and for different FIM and SPM transaction databases. By varying the

[3]https://github.com/zakimjz/IBMGenerator

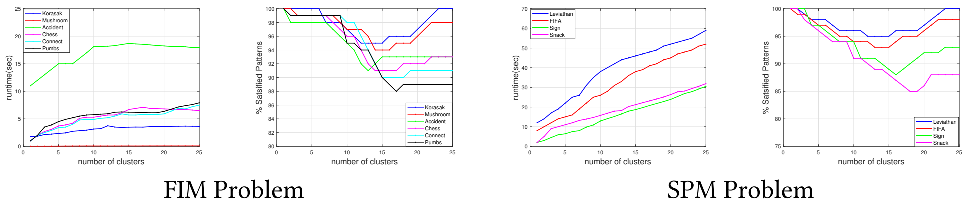
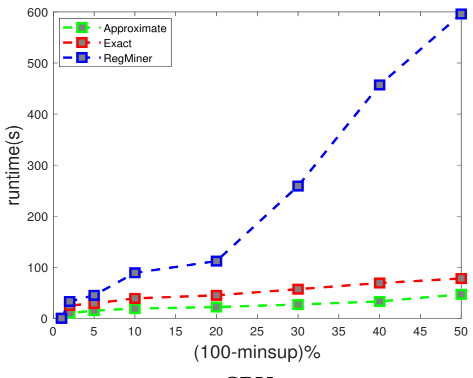


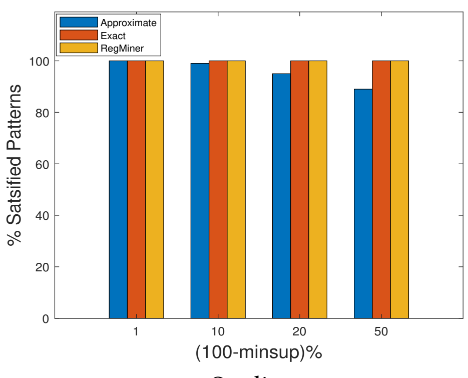
Fig. 13. Runtime (seconds) and percentage (%) of the frequent and sequential patterns of the CBPM framework with different number of clusters.

number of clusters from 1 to 25, the accuracy of our approach exceeds 88% for all cases. However, the results vary from database to database. With smaller number of clusters, fewer separator items are observed, and then high accuracy is obtained. By increasing the number of clusters until a specified value, a higher number of separator items are observed. As a result, the accuracy is reduced. By increasing further the number of clusters, more independent clusters are derived, and then the accuracy is increased up to a certain point. Moreover, we can categorize the transaction databases into two categories, sparse and non sparse data. We can say that the accuracy with non sparse data is better than the accuracy with sparse data. Specifically, the accuracy of Korasak and Mushroom, which are considered as non sparse data, exceeds 94% whatever the case used. However, the accuracy of the sparse data, as the case for the remaining transaction databases, can goes under 90%. This is explained by the fact that with non-sparse data, fewer number of items per transaction is observed. Consequently fewer number of separator items among clusters as compared to the sparse data, which contain higher number of items per transaction, and as a result, high correlation between clusters are derived. In terms of the runtime, while varying the number of clusters from 1 to 25, we can see that the runtime significantly changes with number of clusters, in particular for the Accident transaction database that contains high number of items and transactions. We can explain this as follows. There is an trade off between mining and clustering steps. If we consider few number of clusters, we obtain high number of transactions per cluster. As a result, the clustering step consumes less time than the mining step, and if we consider high number of clusters, we obtain few number of transactions per cluster. Thus, the clustering step consumes more time than the mining step. From these results, we can conclude that our approach is very sensitive to the number of clusters. Choosing the best number of clusters value is a critical issue of our approach. It depends to several factors: the number of items, the number of transactions, and the density of each transaction database. Moreover, when choosing few number of clusters, we obtain a high accuracy, but this is not useful for the parallel approach, where we need more independent clusters. Studying the meta-features of each transaction database and fixing the number of clusters automatically are still open research questions.

Case study: trajectory pattern mining. This experiment aims to show the performance of the proposed framework on real trajectory database called T-Drive [?]. It provides trajectories of 10, 357 different taxis for several days. Each of which is saved in one file. All taxi trajectories are merged to one file providing 68, 872 trajectories. A preprocessing step is performed by transforming each trajectory to one transaction, where all points visited by such trajectory is considered as items in the corresponding transaction. We integrated the CBPM framework with the first phase (Mining Compact Sequential Patterns) of RegMiner algorithm [?]. Figures 14 present the runtime and the percentage of frequent patterns of the original RegMiner with and without using exact and approximate strategy of CBPM framework. The results reveal the stability of RegMiner in terms of runtime performance, when using CPBM framework, this is without losing on the percentage

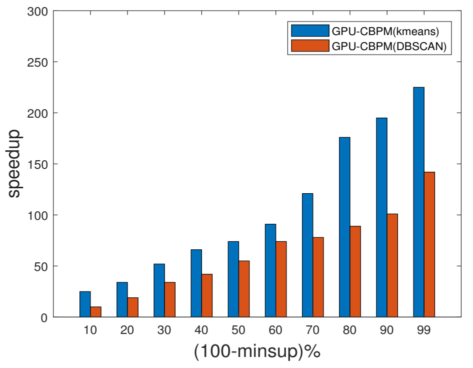


CPU

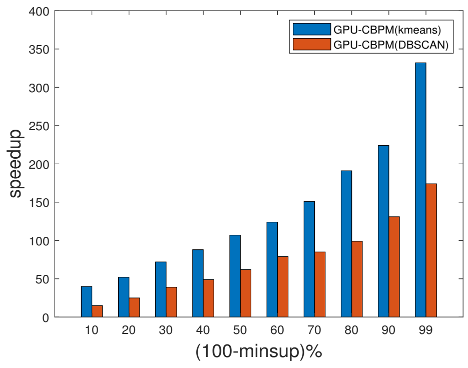


Quality

Fig. 14. Runtime (seconds) and percentage (%) of the frequent patterns of the RegMiner algorithm on T-Drive trajectory database with and without using the CBPM framework.



Frequent Itemset Generator Database



Sequence Generator Database

Fig. 15. Speedup of the GPU-CBPM framework.

Table 4. Percentage of amount of time of the three steps of GPU-CBPM framework.

		GPU-CBPM(Exact)			GPU-CBPM(Approximate)		
Dataset	#Cluster	Decomposition	Mining	Postprocessing	Decomposition	Mining	Postprocessing
Frequent Itemset Generator	2	15	71	14	21	79	0
	5	18	70	12	26	74	0
	10	23	65	12	29	71	0
Sequence Generator	2	12	77	11	20	80	0
	5	15	76	9	27	73	0
	10	19	73	8	29	71	0

of satisfied patterns (up to 89% for all cases). This is explained by the fact only highly correlated trajectories are mined together, instead of exploring the whole T-Drive trajectory database.

6.3 Performance of the parallel version

The GPU-CBPM has been implemented using the CUDA package. Experiments have been carried out on a CPU host coupled with a GPU device. The CPU host is a 64-bit quad-core Intel Xeon

E5520 with a clock speed of 2.27 GHz. The GPU device is an Nvidia Tesla C2075 with 448 CUDA cores (14 multiprocessors with 32 cores each) and a clock speed of 1.15 GHz. It has 2.8 GB of global memory, 49.15 KB of shared memory, and a warp size of 32. Both the CPU and GPU are used in single precision. The parallel version GPU-CBPM is evaluated using the speed up, which is determined by the ratio on runtime of parallel algorithm and the runtime of the serial version. We used the parallel implementation of Zhang's work [**?**] in the mining process for finding the relevant patterns on each cluster. Figures 14 present the speedup of our GPU implementation compared to the serial implementation using IBM Synthetic Data Generator for Itemsets and Sequences to generate 1 million of transactions and 10,000 different items. The use of IBM Synthetic Data Generator allows to generate sparse transactions (transactions with high number of items), very common way to validate parallel approaches on GPU. We also used different ways to decompose the transactions using k-means and DBSCAN algorithms. By varying with the minimum support values from 90% to 1%, the speedup of our GPU implementation increases and reaches 332 for sequence transactions database using k-means algorithm. The results reveal that the speedup on sequence database, and with low minimum support values is more interested than speedup on itemset database, and with high minimum support values. This could be explained by the fact that the parallel implementation performs well on complex pattern mining problem, and with huge search space. Indeed, sequential pattern mining is more complex than frequent itemset mining problem, and setting low minimum support values engenders more number of candidate patterns compared to those generated by setting high number of minimum support values. The results also reveal that the way of decomposing transactions influences on the performance of our GPU implementation. Thus, parallel implementation with k-means highly outperforms DBSCAN scenario in all cases, and whatever the minimum support value. Indeed, with k-means, our GPU implementation reaches speedup of 332, but with DBSCAN, our GPU implementation does not exceed 170. These results are explained by the fact that k-means generates clusters with approximately the same number of transactions, whereas DBSCAN generates clusters with different number of transactions. As result, the load balancing between the GPU blocks using DBSCAN is minimized, and consequently the synchronization cost will be high. This reduces the overall performance of our GPU implementation. Thinking about efficient strategies to reduce the synchronization cost of GPU-CBPM is an open research issue of this work. Another experiment has been carried out to calculate the percentage of amount of time of the three steps included in GPU-CBPM framework by using k-means algorithm in the decomposition step. The results are reported in Table 4, regarding to this table, we can say that the GPU-CBPM spent more time in the mining step for all cases. In addition, when increasing with the number of clusters from 2 to 10, GPU-CBPM consumes much time in the decomposition step, and less time in the mining step. This is due to distributed computing, where high number of clusters have been processed in parallel by the GPU blocks.

7 DISCUSSION AND FUTURE PERSPECTIVES

This section discusses the main findings from the application of the decomposition techniques to the pattern mining problems.

- The first finding of this study is that the proposed framework can deal with big transaction database. This is different from previous pattern mining approaches, which have long execution times, while the whole transaction database is considered in the mining process. The proposed framework is able to not only derive the relevant patterns from the transactions, but also study the different correlation and similarities between the transactions and find out disjoint groups among them. In the context of pattern mining, we argue that considering the decomposition techniques in the preprocessing step allows to quickly derive the relevant patterns.

- From a data mining research standpoint, CBPM is an example of combining data mining techniques. In our specific context, decomposition meets pattern mining for dealing with big transaction databases and boost the mining process. This adaptation is implemented in different phases, such as decomposition, and mining process.
- Another finding of this study is that high-performance computing tools benefits from the data preprocessing by using decomposition. Thus, each node (GPU block in our case) deals with similar transactions, this accelerates the mining process.
- The last observation is that the framework is generic and can be applied in any pattern mining problem, contrary to the other algorithms, which can deal only a particular pattern mining problem. The five pattern mining problems illustrated in this paper are just an example of applications of our framework. Other pattern mining problems such as erasable patterns [**?**], occupancy patterns [**?**] and others may be solved by our framework.

Motivated by the promising results shown in this paper, different directions may be investigated:

(1) **Improving the decomposition step.** HAC, k-means, bisecting k-means and DBSCAN have been used as decomposition techniques. Additional techniques can possibly be used for reducing the number of shared items. Thus, an interesting topics for future work is to integrate other decomposition techniques into the CBPM framework, such as intelligent hierarchical [**?**], overlapping [**?**], or methods from other fields such as entity resolution and/or record linkage [**? ? ? ?**]. Another thing that can be done is to find an appropriate mechanism to automatically fix the number of clusters. Using several runs to find the best value of the number of clusters is not very efficient in practice, even for the GPU-based parallel implementation. One way to address this issue is to create a knowledge base containing each training transaction database, with the best value of the number of clusters, and then study the correlation between the meta-features of the transaction databases (number of items, number of transactions, sparsity value, etc.), and the best values of the number of clusters. This can help to automatically predict the best value of the number of the clusters of the new transaction database.

(2) **Improving the mining step.** We plan to boost the performance of the CBPM and apply it to big data mining applications by exploiting other high-performance computing tools such as cluster computing [**?**]. In this context, strategies to deal with load balancing are inmportant. One way to address this issue is to develop decomposition strategies allowing to find out equitable clusters in terms of number of transactions per cluster. Another way is to develop new strategies for repairing clusters to find clusters with approximately the same number of transactions. Applying CBPM on MapReduce is also an alternative approach for improving the mining step. Performing the partitioning of transactions as pre-processing, and not in the mapping stage, may address the drawbacks of FiDoop-DP algorithm [**?**].

(3) **Case studies.** We already show in this paper a case study of an application of CBPM in the trajectory mining. Motivated by the promising results shown in this first case study, we plan to extend CBPM for solving domain-specific complex problems requiring the mining of big data. This can be found, for instance, in the context of business intelligence applications [**?**] or in the context of mining financial data [**?**]. In particular, runtime performance can be particularly critical in automated trading applications where profits are often made exploiting volatility of share values or currency rates in extremely short time intervals. In these cases, pattern mining algorithms able to discover relevant patterns extremely quickly is likely to open up new opportunities for more intelligent trading. Other potential use is the mining of

sensor data, notably for realtime applications related to internet of things and cyber-physical systems such as road traffic management and related services [?], energy management in smart buildings and smart grids [?], where the mining process is required to be performed within a very short latency.

8 CONCLUSION

We have introduced a new intelligent pattern mining framework, called clustering-based pattern mining (CBPM). It is shown that, CBPM discovers relevant patterns by studying the correlation between the transaction database. The set of transactions are first partitioned using clustering algorithms, where the high correlated transactions are grouped together. From each cluster of transactions, the pattern mining algorithm is launched to discover the relevant patterns, where two, approximate and exact, strategies have been investigated. The CBPM framework has been studied theoretically and experimentally. From the theoretical perspective, the complexity of CBPM is determined for the most common and recent pattern mining algorithms. The results showed that CBPM reduces the complexity of the pattern mining algorithms in terms of the number of clusters. From the experimental evaluation, the CBPM framework has been integrated in the SPMF tool, where five case studies have been provided, i.e., the FIM, WIM, UIM, HUIM and SPM. The results reveal that by using the CBPM, both the runtime and memory usage have been reduced for all tested algorithms, and for both approximate and exact strategies. Moreover, with the exact strategy, the scalability performance is improved without losing the quality of the returned patterns. However, for the approximate strategy, the scalability is largely improved, but with a small loss in the quality and the number of the returned patterns. Thus, the number of the satisfied patterns is up to 89% for all cases, including a real case study of T-Drive trajectory database. To boost the performance of the CPBM, a GPU-based version of CBPM is investigated. It provides efficient mapping between the GPU-blocks and the clusters of transactions, where each cluster of transactions is handled by one GPU block. The results reveal that our GPU implementation achieves significant speedup of up to 332x on a single GPU.