ParaMiner, A Generic Parallel Pattern Mining Algorithm
LIG Internal Research Report

Benjamin Negrevergne, Alexandre Termier, Marie-Christine Rousset and Jean-François Méhaut

Abstract

Pattern mining is a major area of data mining, with many application domains. To solve this inherently costly task, many ad-hoc highly optimized algorithms have been specifically designed for mining certain kinds of data. We describe a *generic and parallel* pattern mining algorithm (PARAMINER), in which we have generalized specific state-of-the-art optimizations. The genericity of PARAMINER is demonstrated by showing that it captures three different pattern mining problems encountered in real-world applications. Its efficiency is demonstrated by showing that it is competitive with different existing algorithms specifically designed for each of these problems.

1 Introduction

Pattern mining is one of the major areas of data mining. Its goal is to discover patterns hidden in large volumes of data, usually by counting their number of occurrences. Since the pioneering work [Agrawal and Srikant, 1994] for mining itemsets in transactional data, pattern mining has been extended to numerous problems such as mining sequences, trees and graphs. Pattern mining has many industrial and scientific applications in various domains such as web usage mining, bioinformatics or drug design. Pattern mining is inherently a costly task for which many ad-hoc highly optimized algorithms have been designed. Such algorithms exploit the specificity of the data to be mined and of the kinds of patterns that are looked for. However, their lack of genericity makes their use difficult for data owners non specialist in pattern mining.

Until recently, a real theoretical foundation for pattern mining was lacking. The main advances made in [Boley *et al.*, 2010; Arimura and Uno, 2009] have been to formalize the link between pattern enumeration and set theory. They have highlighted that for most pattern mining problems, the solutions form a *set system*. They also showed that when the formed set system meets some properties (*strong accessibility* in [Boley *et al.*, 2010], *accessibility* in [Arimura and Uno, 2009]), one can enumerates all the patterns in output-polynomial time. As a proof, they provide algorithms to do so. Enumeration is a critical issue in any pattern mining algorithm, with their contribution, Boley and Arimura have provided an efficient enumeration strategy that can be used in most pattern mining algorithms.

Building on the principles of enumeration in strongly accessible set systems we have designed and implemented PARAMINER, the first *generic and parallel* pattern mining algorithm. We showed that enumeration of patterns occurring in a dataset can be highly optimized in order to achieve practical efficiency. We demonstrate its genericity and its efficiency by applying it to quite different pattern mining problems such as mining gradual itemsets ([Ayouni *et al.*, 2010]) and mining connected relational graphs ([Yan *et al.*, 2005]).

Section 2 provides basic notions for the problem statement. Section 3 describes the PARAMINER algorithm. Section 4 is dedicated to experiments. We conclude in Section 5.

2 Preliminaries and problem statement

Pattern mining is the problem of discovering patterns of interest in a given dataset. Following [Boley *et al.*, 2010; Arimura and Uno, 2009] we define the problem of pattern mining as an enumeration problem in a *set system*. This allows to exploit structural properties of set systems to design efficient enumeration algorithms. In addition to [Boley *et al.*, 2010; Arimura and Uno, 2009], we explicitly involve in the picture the dataset to be mined. In this section, we first set the generic definitions of datasets and patterns that we will use in the rest of the paper. Then, we recall the notions of (strongly) accessible set systems introduced by [Boley *et al.*, 2010; Arimura and Uno, 2009] and we explain how they can impact the efficiency of pattern enumeration.

2.1 Datasets and (closed) patterns

Datasets: In most practical pattern mining problems, the *dataset* $\mathcal{D}_E$ to be mined can be defined over a finite *ground set* E as a multiset of subsets of E. Each element of $\mathcal{D}_E$ is called a *transaction*.

For instance, in the context of market basket analysis [Agrawal and Srikant, 1994], the ground set $E = \{a, b, c, \dots, \}$ is a set of items identifying products such as $apple$, $beer$, $chocolate$, and each purchase ticket stored in the so-called transactional database is an itemset, i.e., a subset of products. In our setting, the transactional database will be represented by the corresponding dataset $\mathcal{D}_E = \{ab, abc, ab\}$, where we will denote the itemset $\{a, b, c\}$ by abc as a shortcut.

In the context of gene network analysis [Yan *et al.*, 2005], the ground set E is the set $G \times G$ of pairs of genes representing all the possible interactions between a given set G of genes, and the dataset is a particular multiset of connected graphs (i.e., subsets of $E = G \times G$) modeling gene networks that are extracted from experimental observations.

Patterns: given a ground set E, a *pattern* is a subset of E *occurring* in the dataset and verifying a *property of interest* $IntProp$. A pattern P *occurs* in $\mathcal{D}_E$ if it is a subset of at least one transaction of the dataset.

Support set of a pattern: the *support set* $\mathcal{D}_E[P]$ of a pattern P is the multiset of transactions of $\mathcal{D}_E$ in which it occurs.

Property of interest: It varies depending on the application needs. It may depend on the dataset or not. For instance, in many pattern mining applications, the property of interest is based on a frequency property: the searched patterns are those that occur frequently (or rarely) in the dataset. In other application settings however, the property of interest is not related to the dataset but to the patterns themselves and the ground set on which they are defined. For example, the searched patterns may be those including a given element of E, or for graph patterns those of a given size or those that are connected. In this paper, we will consider properties of interest so that the interest of a pattern, if it depends on the dataset, can be checked on its support set: $IntProp(P, \mathcal{D}_E) = IntProp(P, \mathcal{D}_E[P])$. This constraint is not very restrictive since it is satisfied by the properties of interest in most pattern mining applications, in particular those based on frequency.

Closed patterns: a pattern P is *closed* if there does not exist any strict subset of P that is also a pattern with the same support set.

Closure operator: Applied to a pattern P, it computes the maximal patterns (w.r.t. the set inclusion) having the same support set as P. By construction, the closure of a pattern is closed. Note that if the property of interest satisfies the above constraint, the closure of any pattern can be computed from the support set of that pattern. Although this is not guaranteed in general, the closure of a pattern is unique in most of the pattern mining applications (e.g., those based on frequency). The set of closed patterns represents a lossless compression of the set of patterns occurring in a given dataset. The pattern mining problem that we consider can now be defined.

Pattern mining problem: given a ground set E, a dataset $\mathcal{D}_E$ and a property of interest $IntProp$, output the closed patterns occurring in $\mathcal{D}_E$. A naive solution to compute the complete set of solutions is to enumerate all the subsets of E and to test for each of them if *(i)* they occur in the data, *(ii)* they verify the property of interest (i.e. they are patterns), and *(iii)* they are closed. This solution is not feasible in practice, due to the size of the powerset of E. Most pattern mining algorithms use an enumeration strategy which performs a constructive enumeration of the closed patterns, using a pattern *augmentation* relation.

Pattern augmentation: A pattern Q is an *augmentation* of a pattern P if there exists $e \in E$ such that $Q = P \cup \{e\}$.

Recent works [Boley *et al.*, 2010; Arimura and Uno, 2009] have shown that the structural properties of the augmentation relation depend on the *accessibility* properties of the *set system* formed by the patterns.

2.2 Set systems and accessibility properties

A *set system* over a ground set E is a pair $(E, \mathcal{F})$, where $\mathcal{F} \subseteq 2^E$ is a family of subsets of E. In the context of pattern mining, $\mathcal{F}$ is the set of patterns that is intentionally defined by the property of interest.

The following definitions define three accessibility properties, from the least strong to the strongest one (as stated in Proposition 1).

Definition 1 (Accessible set system) *A set system $(E, \mathcal{F})$ is* accessible *if for every non-empty $X \in \mathcal{F}$, there exists some $a \in X$ such that $X \setminus \{a\} \in \mathcal{F}$.*

Definition 2 (Strongly accessible set system) *A set system $(E, \mathcal{F})$ is* strongly accessible *if for every $X, Y \in \mathcal{F}$ with $X \subset Y$, there exists some $a \in Y \setminus X$ such that $X \cup \{a\} \in \mathcal{F}$.*

Definition 3 (Independent set system) *A set system $(E, \mathcal{F})$ is* independent *if $Y \in \mathcal{F}$ and $X \subseteq Y$ together imply $X \in \mathcal{F}$.*

Proposition 1 (Relationship between accessibility properties) *Let $(E, \mathcal{F})$ be a set system:*
- *if it is independent, then it is strongly accessible,*
- *if it is strongly accessible, then it is accessible.*

When the set system formed by the patterns is accessible, there is a way to reach each pattern by repeatedly augmenting patterns, starting from the empty set (denoted $\perp$). The set of patterns together with the augmentation relation form a strict partial order with $\perp$ as its minimum element, thus having a directed acyclic graph (DAG) structure. In order to avoid producing several times the same (intermediate) patterns, most pattern mining algorithms build iteratively an **enumeration tree** that is in fact a *covering tree* of that DAG. An important source of limitation in practice is the space required for storing the enumeration tree under construction because its size grows exponentially in function of the size of the ground set E. The problem of designing poly-space algorithms for enumerating closed patterns in accessible set systems has been investigated recently ([Boley *et al.*, 2010; Arimura and Uno, 2009]). We summarize now their underlying principles that we have re-used to design our PARAMINER algorithm.

2.3 Principles of poly-space closed pattern enumeration algorithms

Given a ground set E, a dataset $\mathcal{D}_E$, a property defining the patterns of interest, and a closure operator, poly-space pattern enumeration algorithms compute the set of closed patterns occurring in $\mathcal{D}_E$ *using a polynomial space*. They generate new closed patterns by *augmentation* and *closure* of a current pattern ($\perp$ at the beginning of the process). Without storing the whole enumeration tree under construction, a core issue is to detect whether the new patterns have already been produced in another branch that is no longer in memory.

First principle: An order on E is chosen, and patterns and transactions (that are subsets of E) are represented as ordered sequences of elements of E. For instance, if E is made of the elements b, a, c, and d, a possible order is $a < b < c < d$, and the subsets $\{b, a\}$ and $\{b, d, a\}$ will be represented by the sequences $[a, b]$ and $[a, b, d]$ (for which we will use the notation ab and abd as a shortcut). The empty set is denoted

by $\perp$. The **lexicographic order** induced by the order defined on E is used to compare the patterns, for example: $ab < abd$.
Second principle: For a given closed pattern P ($\perp$ at the beginning), generate all its augmentations $P \cup \{e\}$ such that $P < P \cup \{e\}$ and, if they occur in the dataset and verify the property of interest, compute their closure Q. Iterate the process on those patterns Q for which P is the **first parent**.

Definition 4 (First parent) *Let P be a closed pattern, and Q the closure of an augmentation $P \cup \{e\}$ of P such that $P < P \cup \{e\}$. P is the* first parent *of Q if there does not exist a closed pattern $P' < P$ and an element e' such that $P' < P' \cup \{e'\}$ and Q is the closure of $P' \cup \{e'\}$.*

When the set formed by the patterns is *accessible*, testing that a given closed pattern P is the first parent of a closed pattern Q is a difficult task that may lead to a reverse generation of a whole branch of the enumeration tree (see [Arimura and Uno, 2009]). On the other hand, when the set system formed by the pattern is *independent*, checking whether P is the first parent of Q can be reduced to check whether $Q > P$ ([Boley *et al.*, 2007]). When the set system formed by the patterns is *strongly accessible*, this test is more complicated but much simpler than for the case of simply accessible systems. It has been shown in [Boley *et al.*, 2010] that it can be done efficiently by remembering the list of augmentations performed on the branch ending to P, and checking if elements of these augmentations appear in Q. This list can be encoded in an **exclusion list** of elements of E, the size of which is in $O(log(|E|))$. As few real-world pattern mining problems correspond to independent set systems, we focus on problems corresponding to *strongly accessible* set systems.

3 Generic parallel pattern mining algorithm

In this section, we present the PARAMINER algorithm, We first explain how it exploits parallelism in the construction of the enumeration tree. Then, we show how we have generalized state-of-the art optimizations for dataset reduction.

3.1 Parallel enumeration of the closed patterns

The enumeration strategy implemented in PARAMINER is based on the principles recalled in Section 2.3 for the strongly accessible systems. They are well-suited to the parallel and independent exploration of each branch of the enumeration tree just by handling an **exclusion list** of elements. The exploration of a branch is performed by the recursive $expand$ procedure described in Algorithm 2. It takes as input a closed pattern P, the reduced dataset $\mathcal{D}_P$ associated to it (see Section 3.2), and an exclusion list EL that holds for all the branches issued from P. It expands the pattern P by augmenting it with elements e occurring in its support set (Line 1). If those $P \cup \{e\}$ satisfy the property of interest $IntProp$ (given as input to PARAMINER), it applies the appropriate *closure operator* Clo (given as input to PARAMINER) to get closed patterns Q. For each of them, Line 4 checks whether P is their **first parent** by a simple intersection test with the exclusion list EL (as in [Boley *et al.*, 2010]). Those Q are outputted as soon as they are produced in Line 6. The $expand$ procedure is then recursively called (Line 8) on each Q, together with its corresponding reduced dataset $\mathcal{D}_Q$ (computed Line 7 by the $reduce$ function described in Algorithm 3, Section 3.2), and the same exclusion list EL.

In Algorithm 2, parallelism is expressed by the **spawn** directive of Line 8. Instead of directly executing the recursive calls to $expand$, these calls are pushed into a waiting list of parallel jobs. As soon as a thread becomes idle, it can take any of these waiting $expand$ calls and develop this node of the enumeration tree. This avoids load unbalance: if the enumeration tree has a very big subtree, many $expand$ calls will be spawned there, and more threads will end up processing this subtree. In practice, **spawn** is implemented with the Melinda library, that has been used to implement a parallel version of LCM [Negrevergne *et al.*, 2010]. Melinda preserves cache locality as much as possible by assigning in priority to a thread the recursive calls that it has spawned. PARAMINER is described in Algorithm 1: it simply consists in executing the $expand$ procedure applied to the empty closed pattern $\perp$, the whole dataset and an empty exclusion list.

Algorithm 1 PARAMINER

Require: groundset E, interest property $IntProp$, closure operator Clo, dataset $\mathcal{D}$
Ensure: Output all closed patterns occurring in $\mathcal{D}$.
1: $expand(\perp, \mathcal{D}, \emptyset)$

Algorithm 2 Expanding a closed pattern P

Require: P, Reduced dataset $\mathcal{D}_P$, Exclusion List EL
Ensure: Output all closed patterns issued from P occurring in $\mathcal{D}_P$.
1: **for all** e such that $\mathcal{D}_P[\{e\}] \neq \emptyset$ **do**
2: **if** $IntProp(P \cup \{e\}, \mathcal{D}_P)$ **then**
3: $Q := Clo(P \cup \{e\}, \mathcal{D}_P)$
4: **if** $EL \cap Q = \emptyset$ **then**
5: *//P is Q's the first parent*
6: **output** Q
7: $\mathcal{D}_Q := reduce(\mathcal{D}_P, e, EL)$
8: **spawn** $expand(Q, \mathcal{D}_Q, EL)$
9: $EL := EL \cup \{e\}$
10: **end if**
11: **end if**
12: **end for**

3.2 Dataset reduction

A key point for limiting the complexity of PARAMINER is to reduce the dataset involved in the computations of the property of interest $IntProp$ (Line 2 of Algorithm 2) and of the closure function Clo (Line 3 of Algorithm 2). For each closed pattern Q produced in the $expand$ procedure, a new dataset is constructed, that is restricted to transactions that are sufficient to compute further augmentations of Q. This *reduced dataset* is denoted $\mathcal{D}_Q$ and is constructed in line 7 of Algorithm 2 by calling the $reduce$ function applied to *(i)* the reduced dataset $\mathcal{D}_P$ of the first parent P of Q, *(ii)* the element e with which P has been augmented to produce Q (by closure of $P \cup \{e\}$), and *(iii)* the exclusion list EL. The dataset reduction is described

in Algorithm 3. It first removes transactions from $\mathcal{D}_P$, and then suppresses elements from the remaining transactions.

Removing transactions: It is done in Line 1 of Algorithm 3, and consists simply in initializing $\mathcal{D}_Q$ with the support set $\mathcal{D}_P[\{e\}]$ of $\{e\}$ in $\mathcal{D}_P$, where e is the element chosen to augment the pattern P such that Q is the closure of $P \cup \{e\}$. This is not a limitation for correctly checking $IntProp$ because $IntProp(Q, \mathcal{D}_E) = IntProp(Q, \mathcal{D}_P[\{e\}])$ for the following reasons:
- by the constraint on $IntProp$ imposed in Section 2: $IntProp(Q, \mathcal{D}_E) = IntProp(Q, \mathcal{D}_E[Q])$,
- by definition of the closure: $\mathcal{D}_E[Q] = \mathcal{D}_E[P \cup \{e\}]$ (= $\mathcal{D}_E[P][\{e\}]$),
- $IntProp(Q, \mathcal{D}_E[P][\{e\}]) = IntProp(Q, \mathcal{D}_P[\{e\}])$, since the transactions in $\mathcal{D}_P$ are the same transactions as $\mathcal{D}_E[P]$ (with some non-significant elements removed, see below).

Suppressing elements from transactions: The elements of the exclusion list EL that can be safely suppressed from transactions in $\mathcal{D}_Q$ are the elements that are guaranteed not to appear in any closure of a pattern (i.e. that would disappear after the closure done in Line 3 of Algorithm 2). For determining quite easily such elements, $\mathcal{D}_Q$ is partitioned (by the function $partition(\mathcal{D}_Q, EL)$ called Line 3 in Algorithm 3) in groups of sets of transactions that have the same elements except elements of EL. For each group G of the partition, we suppress from each of its transactions the elements of EL that do not appear in all the transactions of the group. Such elements will not belong to the closure of any pattern Q' further produced from augmentations of Q and supported by the transactions in G. This is done in Lines 4–8 of Algorithm 3. Note that this is a generalization of the so-called *prefix intersection* optimization at the core of LCM [Uno *et al.*, 2004].

Algorithm 3 The dataset reduction algorithm

Require: Reduced dataset $\mathcal{D}_P$ of a pattern P, Element e such that $Q = Clo(P \cup \{e\}, \mathcal{D}_P)$, Exclusion List EL
Ensure: Reduced dataset of Q: $\mathcal{D}_Q$
1: $\mathcal{D}_Q \leftarrow \mathcal{D}_P[\{e\}]$
2: *// Suppress elements from transactions in* $\mathcal{D}_Q$
3: **for all** $G \in partition(\mathcal{D}_Q, EL)$ **do**
4: **for all** $e \in EL$ **do**
5: **if** there exists $t' \in G$ such that $e \notin t'$ **then**
6: **for all** $t \in G$ **do**
7: $t' \leftarrow t \backslash \{e\}$
8: $\mathcal{D}_Q \leftarrow \mathcal{D}_Q \backslash \{t\} \cup \{t'\}$
9: **end for**
10: **end if**
11: **end for**
12: **end for**
13: **return** $\mathcal{D}_Q$

We have implemented PARAMINER in C++. We do not detail here other optimizations though they are important for the performances reported in Section 4. They are based on compact data structures for representing the datasets and for efficiently computing support sets of elements, in the spirit of *occurrence deliver* in LCM [Uno *et al.*, 2004].

4 Experiments

We have tested PARAMINER on three different pattern mining problems for which we only had to implement the property of interest and the closure operator. Section 4.1 shows the encoding of these different problems in (independent or strongly accessible) set systems. Section 4.2 summarizes experimental results on real-world data.

4.1 Genericity of our approach

We present now the different problems and the corresponding ground sets, datasets, properties of interest and closure operators.

Mining closed frequent itemsets: For this reference problem, first defined by [Pasquier *et al.*, 1999], the ground set E and the dataset $\mathcal{D}_E$ have been introduced in Section 2.1.
Property of interest: For $P \subseteq E$, $IntProp(P) \equiv |\mathcal{D}_E[P]| \geq \varepsilon$ (for a given constant ε).
Closure operator: $Clo(P) = \bigcap_{t \in \mathcal{D}_E[P]} t$.
Accessibility: [Boley *et al.*, 2010] have shown than the associated set system is *independent* and thus *strongly accessible*.

Mining closed frequent connected relational graphs: A relational graph is a labelled graph in which all the node labels are distinct. Such graphs can represent gene networks as well as social networks [Yan *et al.*, 2005]. The corresponding ground set and dataset have been introduced in Section 2.1.
Property of interest: For $G \subseteq E$, $IntProp(G)$ if 1) $|\mathcal{D}_E[G]| \geq \varepsilon$ (for a given constant ε) and 2) G is connected.
Closure operator: The closure of a pattern P is the intersection of the connected graphs of $\mathcal{D}_E[P]$ containing P.
Accessibility: The set system associated to this problem is *strongly accessible*, as a direct consequence of the strong accessibility of the set system associated with Problem 5 in [Boley *et al.*, 2010].

Mining closed frequent gradual itemsets: This challenging recent problem consists in mining attributes co-variations in numerical databases [Ayouni *et al.*, 2010]. Consider the example database of Table 1.

Place	Temperature in °C	Electric consumption in W
p_1	0	2000
p_2	10	1000
p_3	20	500
p_4	30	1500

Table 1: Example numerical dataset

When considering the sequence of the tuples p_1, p_2 and p_3, it appears that an increase in temperature is correlated with a decrease in electric consumption. A pattern is the *gradual itemset* $(Temperature^{\uparrow}, ElectricConsumption^{\downarrow})$ which is *respected* by the sequence of tuples $[p_1, p_2, p_3]$. Note that symmetrically, $(Temperature^{\downarrow}, ElectricConsumption^{\uparrow})$ is respected by $[p_3, p_2, p_1]$.

Let $A = \{a_1, ..., a_m\}$ be the set of attributes and $\mathcal{DB} = \{p_1, ..., p_n\}$ be a set of tuples over A. The problem of mining closed frequent gradual itemsets can be represented in our framework in the following way:

Ground set: E is the set of attributes variations: $E = \{a_1^{\uparrow}, a_1^{\downarrow}, ..., a_m^{\uparrow}, a_m^{\downarrow}\}$.
Dataset: $\mathcal{D}_E$ is the set of transactions $t_{i,j}$ (for all $i, j \in [1..n], i \neq j$) such that $t_{i,j}$ contains $a^{\uparrow}$ if $p_i[a] < p_j[a]$, and $a^{\downarrow}$ otherwise, for every attribute $a \in A$ ($p[a]$ denoting the value of attribute a for tuple p). Among the transactions of our example, $t_{1,2} = \{Temperature^{\uparrow}, ElectricConsumption^{\downarrow}\}$ and $t_{3,4} = \{Temperature^{\uparrow}, ElectricConsumption^{\uparrow}\}$.
Property of interest: $G = \{a_{x_1}^{v_1}, ..., a_{x_k}^{v_k}\}$ is a pattern if it is contained in at least ε transactions of $\mathcal{D}_E$ (where ε is a given constant) forming a *path*. A sequence of transactions $[t_{i_1,j_1}, ..., t_{i_q,j_q}]$ form a path if $\forall x \in [2, q]\ i_x = j_{x-1}$.
$(Temperature^{\uparrow}, ElectricConsumption^{\downarrow})$ is contained in the two transactions $t_{1,2}$ and $t_{2,3}$ that form a path.
Closure operator: The closure of a pattern P is the intersection of the transactions in $\mathcal{D}_E[P]$.
Accessibility: The corresponding set system is *independent*. However, avoiding mining symmetrical patterns is crucial for gaining efficiency. In fact, the problem of mining closed frequent gradual itemsets dealt with in the literature (and in our experiments) is a variant in which only half of the patterns are enumerated without loss of information. We have shown that the associated set system is not independent anymore but is still *strongly accessible* (proof omitted due to lack of space).

4.2 Experimental results

Itemsets: In this section, we first present the performance evaluation of PARAMINER for mining closed frequent itemsets. The execution times are compared with the state-of-the-art parallel ad-hoc algorithms PLCM [Negrevergne *et al.*, 2010] and MT-Closed [Lucchese *et al.*, 2007]. The results are reported for the sparse dataset *BMS-WebView* and the dense dataset *Accidents* in Figure 1. These two datasets have been selected from [Goethals, 2003].

The computing platform is based on four Intel Xeon X7560 processor (2.27 GHz) with eight cores, for a total of 32 cores. Each processor has a shared cache of 24 MB. There is a total of 64 GB of main memory.

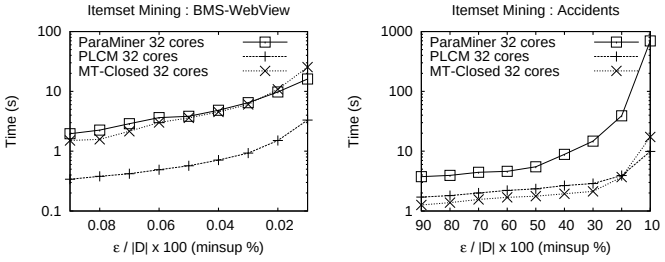

Figure 1: Itemsets

On the sparse dataset, PARAMINER and MT-CLOSED have very close performance. Performance of PARAMINER is one order of magnitude slower than PLCM. On the dense dataset, PARAMINER is between one and two orders of magnitude slower than both PLCM and MT-CLOSED.

PLCM and MT-CLOSED can exploit the problem specificity by ignoring infrequent elements both in enumeration and for more aggressive dataset reduction. Although PARAMINER cannot make these assumptions for the sake of genericity, it still exhibits reasonable run times. It is worth noticing that for the same datasets, any implementation of Apriori is two to three orders of magnitude slower than PARAMINER.

Graphs: The problem of mining closed frequent connected relational graphs is studied in [Yan *et al.*, 2005] but no free implementation is available. Therefore we only provide performance results and execution times for PARAMINER. We use a real gene network dataset with 1000 graphs, each having 300 genes. The results are presented in Figure 2a. These results show that PARAMINER can mine this real-world dataset within minutes, which is enough in practice.

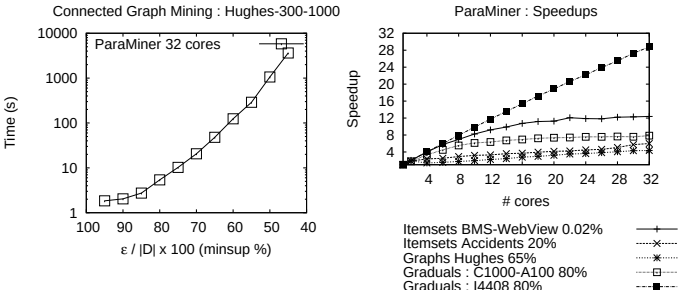

Figure 2: a) Relational graphs b) Speedups

Gradual itemsets: We compare PARAMINER with the state-of-the-art parallel algorithm PGLCM [Do *et al.*, 2010], and with PGP-MC [Laurent *et al.*, 2010]. PGP-MC only mines frequent gradual itemsets (not necessarily closed). We use two datasets: *C1000A100* a synthetic dataset with 1000 tuples and 100 attributes, and *I4408*, a real gene expression dataset with 100 tuples and 4408 attributes. The results are shown in Figure 3.

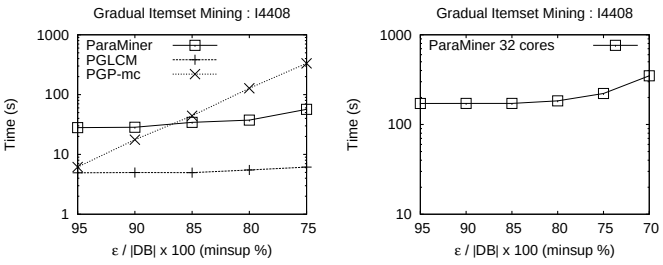

Figure 3: Gradual itemsets

On the synthetic dataset, PARAMINER is one order of magnitude slower than PGLCM. As expected, with lower support values, it is more efficient than PGP-MC. On the real-world dataset, the optimizations implemented in

PARAMINER are crucial to scale up with many attributes. This allows PARAMINER to be two orders of magnitude faster than PGLCM that needs more than ten hours of computation and is not represented an Figure 3. For this dataset, PGP-MC cannot complete due to memory exhaustion.

We show in Figure 2b the speedups for parallel executions of PARAMINER in the previous experiments. Speedup is defined by the ratio T_1/T_p between execution time on one core (T_1) with the execution time on p cores (T_p). Depending on problems and datasets, the figure presents varied results. PARAMINER shows better speedups when there are fewer patterns and each pattern needs complex $IntProp$ computations. This absorbs the cost of computing reduced datasets. This is the case when mining gradual itemsets in *I4408*. On the other hand, PARAMINER doesn't scale up as well on synthetic datasets that are typically more regular and thus have more patterns. Moreover, on simple problems such as mining closed frequent itemsets, computations for $IntProp$ and Clo are relatively simple, which means that the delay between the construction of two reduced matrices is very small. In such cases, the limiting factor is no longer computing power but memory bandwidth. The number of core is increasing more quickly than the memory bandwidth. When a data is not stored in the cache, the processor has to wait for data transfer from the memory, leading to lower speedups.

5 Conclusion

We have presented PARAMINER, a generic and parallel algorithm adapted to pattern mining problems corresponding to strongly accessible underlying set systems. We have shown that PARAMINER can be used to mine real data with runtimes comparable to state-of-the-art ad-hoc algorithms.

To the best of our knowledge, PARAMINER is the first *parallel* generic pattern mining algorithm. Data Mining Template Library (DMTL) [Chaoji *et al.*, 2008] and iZi [Flouvat *et al.*, 2009] provide implemented generic pattern mining algorithms. DMTL is restricted to properties of interest based on frequency counting, and is composed of ad-hoc algorithms to mine itemsets, sequences, trees and graphs. iZi offers more flexibility in the choice of the property of interest, but cannot tackle sequences, trees or graphs. The experiments in [Chaoji *et al.*, 2008; Flouvat *et al.*, 2009] show that both iZi and DMTL are at least one order of magnitude slower than ad-hoc algorithms for mining *all* frequent patterns, for example efficient implementations of Apriori for itemsets. In contrast, PARAMINER is two to three orders of magnitude faster than such ad-hoc algorithms, thus being more than three orders of magnitude faster than iZi or DMTL.

We plan to improve the parallel performance of PARAMINER by improving its memory bandwidth usage. We also plan to handle other pattern mining problems within our setting (e.g. sequences or trees) in order to provide an Open-Source package with PARAMINER, properties of interest and closure operators for most common pattern mining problems.

References

[Agrawal and Srikant, 1994] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[Arimura and Uno, 2009] Hiroki Arimura and Takeaki Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *SDM*, pages 1087–1098, 2009.

[Ayouni *et al.*, 2010] Sarra Ayouni, Anne Laurent, Sadok Ben Yahia, and Pascal Poncelet. Mining closed gradual patterns. In *ICAISC*, pages 267–274, 2010.

[Boley *et al.*, 2007] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Efficient closed pattern mining in strongly accessible set systems. In *Mining and Learning with Graphs (MLG)*, 2007.

[Boley *et al.*, 2010] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Listing closed sets of strongly accessible set systems with applications to data mining. *Theor. Comput. Sci.*, 411(3):691–700, 2010.

[Chaoji *et al.*, 2008] Vineet Chaoji, Mohammad Al Hasan, Saeed Salem, and Mohammed Javeed Zaki. An integrated, generic approach to pattern mining: data mining template library. *Data Min. Knowl. Discov.*, 17(3):457–495, 2008.

[Do *et al.*, 2010] Trong Dinh Thac Do, Anne Laurent, and Alexandre Termier. Pglcm: Efficient parallel mining of closed frequent gradual itemsets. In *ICDM*, pages 138–147, 2010.

[Flouvat *et al.*, 2009] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. The izi project: Easy prototyping of interesting pattern mining algorithms. In *PAKDD Workshops*, pages 1–15, 2009.

[Goethals, 2003] Bart Goethals. Fimi repository website. `http://fimi.cs.helsinki.fi/`, 2003.

[Laurent *et al.*, 2010] Anne Laurent, Benjamin Négrevergne, Nicolas Sicard, and Alexandre Termier. Pgp-mc: Towards a multicore parallel approach for mining gradual patterns. In *DASFAA (1)*, pages 78–84, 2010.

[Lucchese *et al.*, 2007] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Parallel mining of frequent closed patterns: Harnessing modern computer architectures. In *ICDM*, pages 242–251, 2007.

[Negrevergne *et al.*, 2010] Benjamin Negrevergne, Alexandre Termier, Jean-Francois Mehaut, and Takeaki Uno. Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses. In *HPCS*, pages 521–528, 2010.

[Pasquier *et al.*, 1999] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416, 1999.

[Uno *et al.*, 2004] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, pages 16–31, 2004.

[Yan *et al.*, 2005] Xifeng Yan, Xianghong Jasmine Zhou, and Jiawei Han. Mining closed relational graphs with connectivity constraints. In *ICDE*, pages 357–358, 2005.