Noname manuscript No.
(will be inserted by the editor)

ParaMiner: A generic pattern mining algorithm for multi-core architectures

Benjamin Negrevergne · Alexandre Termier · Marie-Christine Rousset · Jean-François Méhaut

Received: date / Accepted: date

Abstract In this paper, we present ParaMiner which is a *generic* and *parallel* algorithm for closed pattern mining.

ParaMiner is built on the principles of pattern enumeration in strongly accessible set systems. Its efficiency is due to a novel *dataset reduction* technique (that we call *EL-reduction*), combined with novel technique for performing dataset reduction in a parallel execution on a multi-core architecture.

We illustrate ParaMiner's genericity by using this algorithm to solve three different pattern mining problems: the frequent itemset mining problem, the mining frequent connected relational graphs problem and the mining gradual itemsets problem.

In this paper, we prove the soundness and the completeness of ParaMiner. Furthermore, our experiments show that despite being a generic algorithm, ParaMiner can compete with specialized state of the art algorithms designed for the pattern mining problems mentioned above. Besides, for the particular problem of gradual itemset mining, ParaMiner outperforms the state of the art algorithm by two orders of magnitude.

Keywords Data mining · closed pattern mining · parallel pattern mining · multi-core architectures

B. Negrevergne
LIG laboratory, University of Grenoble, France
E-mail: Benjamin.Negrevergne@imag.fr

A. Termier
E-mail: Alexandre.Termier@imag.fr

M.-C. Rousset
E-mail: Marie-Christine.Rousset@imag.fr

J.-F. Méhaut
E-mail: Jean-Francois@imag.fr

1 Introduction

Pattern mining is one of the major areas of data mining. Its goal is to extract patterns hidden in large volumes of data, usually by counting their number of occurrences. Since the pioneering work by Agrawal and Srikant (1994) for mining itemsets in transactional data, pattern mining has been extended to numerous tasks such as mining sequences, trees and graphs. Pattern mining has many industrial and scientific applications in varied domains such as web usage mining, bioinformatics or drug design.

Pattern mining is a costly task due to two main reasons: the size of the search space and the inherent cost required to access large datasets. In order to achieve practical efficiency, many dedicated algorithms have been designed. These algorithms exploit the specificities of the pattern mining problem at hand to reduce the number of candidate patterns generated (first problem) and also to reduce the number of accesses to the dataset (second problem). In this paper we name the problem exploring the search space of all candidate patterns (first problem) the *pattern enumeration problem* and problem of reducing the number of accesses to the dataset to test the candidate patterns (second problem) the *pattern testing problem*.

Due to its relative simplicity, most algorithmic advances were achieved on the seminal problem of mining frequent itemsets. Pattern enumeration was first improved by Pasquier et al (1999) by focusing on *closed* frequent itemsets, and then by providing a tree based enumeration over the search space of closed frequent patterns with the LCM algorithm by Uno et al (2003). Pattern testing was greatly improved by introducing the concept of *dataset reduction* with the FP-Growth algorithm from Han et al (2000). This algorithm performs a tree based enumeration of the frequent itemsets, and creates for each node of that tree a reduced dataset (called *conditional database* in the original paper) that is tailored to compute only the itemsets of the corresponding enumeration subtree. This technique was improved in the LCM2 algorithm by Uno et al (2004), that won the title of most efficient closed frequent itemset miner at the FIMI'04 workshop (Goethals (2004)), and still is the reference algorithm for efficiency of mining.

Despite their huge performance impact, most of these improvements do not transfer easily to other pattern mining problems. The difficulty of adapting state of the art pattern mining techniques to new pattern mining problems slows down pattern mining research. The challenge is thus to design *generic* pattern mining algorithms while providing state of the art efficiency.

Regarding genericity Arimura and Uno (2009), and Boley et al (2010) have recently proposed to study the problem of closed pattern enumeration as the problem of enumerating sets in a set system. The definition of a pattern relies on a simple predicate called *selection criterion* and their theoretical framework can capture a broad range of pattern mining problems such as frequent itemsets, connected relational graphs, rigid sequences and others, with strong complexity guarantees (output-polynomial in time, polynomial in space). How-

ever they do not address the pattern testing problem, which prevents their contribution to be turned into a practically efficient algorithm.

Our goal is to bridge the gap between generic but practically intractable pattern mining algorithms, and dedicated algorithms that are efficient only for a given problem. We present the ParaMiner algorithm that both covers a broad class of pattern mining problems, and achieves practical efficiency thanks to a novel dataset reduction technique and to harnessing multi-core architectures through parallelism. This result will allow pattern mining practitioners to mine new types of datasets and new patterns with little effort. ParaMiner is available as an open source software on authors' web page.

The originality and efficiency of ParaMiner comes from two main contributions:

- The first contribution is a novel and generic dataset reduction operator called *EL-reduction*. Unlike other dataset reduction operators of the litterature, the EL-reduction is independent of the pattern mining problem considered. In order to define this operator, we restricted the framework proposed by Arimura and Uno (2009) and Boley et al (2010) to a slightly more specialized class of pattern mining problems where the patterns have to occur in a *dataset*. We also impose some limitations on the selection criterion, expressed through a new property called *decomposability* that it has to satisfy. Our framework covers most of the practical pattern mining problems studied in the literature.
- The second contribution is a thorough study of the impact of dataset reduction on the parallel execution of ParaMiner, when using a straightforward parallelisation of a depth first search algorithm. To the best of our knowledge this is the first time that such a study is conducted. We especially focus on *memory bus contention*. The results of this study allow us to distinguish, during the search space exploration, the nodes where a dataset reduction *has to* be perfomed and those where dataset reduction will be inefficient, and to devise a criterion to determine *a priori* if a dataset reduction must be performed or not. We show that such criterion significantly improve parallel scalability.

The paper is organized as follow. In Section 2, we will recall the state of the art in closed pattern enumeration. In this setting it is important that each pattern has a unique closure, we will provide a new property called *supported confluence* and will prove that it guarantees closure uniqueness when verified by a pattern mining problem. In Section 3, we will instantiate three different pattern mining problems with ParaMiner to show its genericity. In Section 4, we will describe the ParaMiner algorithm and our original *EL-reduction* technique for dataset reduction. We will also prove the soundness and completeness of ParaMiner. In Section 5, we will first present a detailed study of the impact of dataset reduction on parallel performance and then present a criterion for selecting the dataset reductions that have to be performed. In Section 6 we will compare ParaMiner with several specialized algorithms on the problems we instantiated, to demonstrate that ParaMiner has competitive execution

times. For recent pattern mining problems, it can outperform state-of-the-art ad-hoc algorithms by two orders of magnitude. In Section 7, we will relate our generic framework for pattern mining to other existing generic approaches. In Section 8, we will conclude and give some research perspectives.

2 Generic framework: definitions and algorithmic background

In this paper we focus on the problem of extracting closed patterns (Pasquier et al, 1999). A closed pattern is a representative pattern of a class of equivalent patterns from which it is possible to derive all the patterns of this class.

The set of closed patterns contains exactly the same information as the complete set of patterns, but there are usually much fewer closed patterns, which permits a faster mining and an easier analysis of the results.

In order to provide a generic definition for closed patterns we extend the work of Boley et al (2010) and (Arimura and Uno, 2009). We rely on a homogeneous representation of the patterns and of the dataset using sets. Thanks to this representation, we are able to state a general problem of closed pattern mining that can capture many specific closed pattern mining problems.

2.1 Preliminaries

We define a dataset as a sequence of transactions over a finite *ground set* of elements.

Definition 1 (Dataset) Given a ground set E, a *dataset* $\mathcal{D}_E$ is a sequence of transactions $[t_1, t_2, \ldots, t_n]$ where each transaction is a subset of the ground set E. The set of transaction indices is called the *tid set* and is denoted $T_{\mathcal{D}_E}$.

We also use the following notations:

- $\mathcal{D}_E(i)$, with $i \in T_{\mathcal{D}_E}$ denotes the transaction t_i in $\mathcal{D}_E$.
- $|\mathcal{D}_E(i)|$ denotes the number of elements in $\mathcal{D}_E(i)$.
- $|\mathcal{D}_E|$ denotes the number of transactions in $\mathcal{D}_E$.
- $||\mathcal{D}_E|| = \sum_{i=1}^{|\mathcal{D}_E|} |\mathcal{D}_E(i)|$ denotes the size of $\mathcal{D}_E$.

Many application datasets can be directly stored in this form. For example, for frequent itemsets mining in the context of market basket analysis, the ground set is the set of all available items (e.g., $E = \{apple, beer, chocolate\}$) and each transaction in the dataset is a set of items purchased together by the same customer. An example of a dataset defined over the previous ground set is: $\mathcal{D}_E = [\{apple, beer, chocolate\},\ \{apple, beer\},\ \{apple, chocolate\}]$.

It is also possible to use this definition to mine genomic datasets. For example in the context of gene network analysis proposed by Yan et al (2005), given a set of genes denoted G, the ground set E is the cartesian product $G \times G$ of pairs of genes representing all the possible gene interactions between the genes of G. In the dataset, each transaction is a set of interactions observed during one experiment. An example is shown in Figure 1.

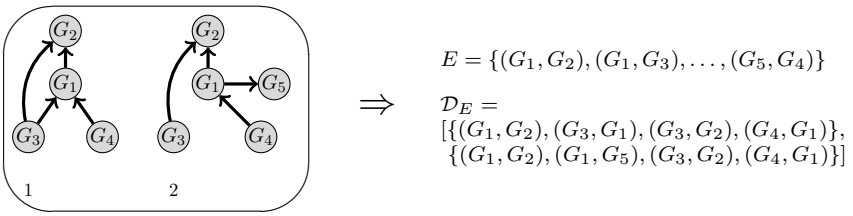

Fig. 1 A dataset of relational graphs, each node is a gene, there is an arc between two genes G_X and G_Y when the gene G_X interacts (i.e. has an influence) with the gene G_Y. The ground set E is the set of all the possible interactions between the set of genes, each transaction in $\mathcal{D}_E$ is a set of interactions between genes.

We now introduce other important definitions:
A *candidate pattern* is defined as a subset of the ground set.

Definition 2 (candidate pattern) A candidate pattern is a subset of the ground set E.

The support set of a candidate pattern is the sub-dataset made of the transactions including this candidate pattern.

Definition 3 (support set) Given a dataset $\mathcal{D}_E$ and a candidate pattern $X \subseteq E$, the support set of X denoted $\mathcal{D}_E[X]$ is the set of transactions in $\mathcal{D}_E$ including X.

In addition, we define the *tid support set* of a candidate pattern X as follow.

Definition 4 (tid support set) The tid support set of a candidate pattern X, denoted $\mathcal{D}_E[\![X]\!]$ is the set of indices of the transactions in $\mathcal{D}_E[X]$: $\mathcal{D}_E[\![X]\!] = \{t \in T_{\mathcal{D}_E} | X \subseteq D_E(t)\}$.

If a candidate pattern $X \subseteq E$ has a non empty tid set in a dataset $\mathcal{D}_E$, we say that X *occurs* in $\mathcal{D}_E$.

A *selection criterion* needs to be provided as a user-specified predicate *Select* in order to discriminate patterns of interest among candidate patterns.

Definition 5 (Pattern) Given a dataset $\mathcal{D}_E$ built over a ground set E, a selection criterion *Select*, a candidate pattern $X \subseteq E$ is a pattern in $\mathcal{D}_E$ if and only if:

1. X occurs in $\mathcal{D}_E$
2. X satisfies the selection criterion in the dataset: $Select(X, \mathcal{D}_E) = true$.

In the following, we denote by $\mathcal{F} \subseteq 2^E$ the set of patterns.

2.2 Closed patterns

Closed patterns were proposed by Pasquier et al (1999) in the context of frequent itemset mining, to reduce the redundancy among the set of the patterns. A pattern is closed if it is the largest pattern occurring in its support set.

Definition 6 (Closed pattern) A pattern $P \in \mathcal{F}$ is *closed* if and only if there exists no strict superset of P that is a pattern in $\mathcal{F}$ with the same support set.

A closure of a pattern is a superset of this pattern with the same support set.

Definition 7 (Closure of a pattern) For a pattern $P \in \mathcal{F}$, a closed pattern $Q \in \mathcal{F}$ is a closure of P if and only if $P \subseteq Q$ and $\mathcal{D}_E[P] = \mathcal{D}_E[Q]$.

This definition guarantees that every pattern admits a closure: Either there is a superset with the same support or the pattern itself is its own closure. Algorithm 1 is a generic algorithm that computes a closure of a pattern by augmenting it with elements from the intersection of the transactions in its support set.

Algorithm 1 A generic closure algorithm

Require: a ground set E, a dataset $\mathcal{D}_E$, a selection criterion *Select* and a pattern P.
Ensure: returns the unique closure Q of P.

$Q \leftarrow P$
$Q_{max} = \bigcap \mathcal{D}_E[P]$ /*Elements included in every transaction in $\mathcal{D}_E[P]$*/
while $\exists e \in Q_{max} \setminus Q : Select(Q \cup \{e\}, \mathcal{D}_E)$ **do**
  /*While there exists e such that $Q \cup \{e\} \in \mathcal{F}$*/
  $Q \leftarrow Q \cup \{e\}$
end while

return Q

The uniqueness of closure is crucial for defining a closure *operator* at the core of closed pattern enumeration algorithms. We exhibit a property called *supported confluence* that is sufficient to guarantee that the closure of every pattern is unique. This property is satisfied if for every two patterns with the same support set, the existence of a non empty pattern in their intersection guarantees that their union is also a pattern.

Property 1 (Supported confluence) Let $\mathcal{F}$ be the set of patterns occurring in a dataset $\mathcal{D}_E$. $\mathcal{F}$ is *support confluent* if and only if:
$Q \cup Q'$ is a pattern for every patterns Q and Q' such that:

i. Q and Q' have the same support set in $\mathcal{D}_E$, $(\mathcal{D}_E[Q] = \mathcal{D}_E[Q'])$
   and
ii. $Q \cap Q'$ includes a non empty pattern, (there exists $Z \subseteq Q \cap Q'$ with $Z \neq \emptyset$ such that $Z \in \mathcal{F}$)

Theorem 1 states that the closure of every pattern is unique if the set of patterns is support confluent.

Theorem 1 (Closure uniqueness)
Let $\mathcal{F}$ be the set of patterns occurring in a dataset $\mathcal{D}_E$. If $\mathcal{F}$ is support confluent, then the closure of every pattern is unique. In this case, we denote $Clo(P, \mathcal{D}_E)$ the closure of P in $\mathcal{D}_E$.

Proof: Suppose that there exists a pattern P that admits two closures denoted Q and Q'. From Definition 7, it is guaranteed that: (1) Q and Q' are patterns, (2) $P \subseteq Q$ and $P \subseteq Q'$, and (3) $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$. From (3), item (i) of supported confluence is verified. Since P is a non empty pattern in $Q \cap Q'$, item (ii) of supported confluence is also verified. Hence $Q \cup Q'$ is also a pattern. Since $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$, $\mathcal{D}_E[Q \cup Q'] = \mathcal{D}_E[Q] \cap \mathcal{D}_E[Q'] = \mathcal{D}_E[Q] = \mathcal{D}_E[Q']$. However, if Q is closed, there exists no strict super set of Q that admits the same support set. Hence $Q = Q \cup Q'$. The same goes for Q', and thus $Q = Q'$. Therefore the closure of any pattern P is unique. □

Other characterizations of the closure uniqueness have been published. In particular the property of *confluence*, introduced by Boley et al (2010), which has inspired the supported confluence. A set of patterns defined over a ground set is confluent if and only if the union of two patterns having a non empty pattern in their intersection is also a pattern. If the confluence property is satisfied for a set of patterns, then the closure uniqueness is guaranteed for *every possible dataset* that can be defined over the same ground set.

Guaranteeing that the closure is unique for a given set of pattern and any dataset is uselessly restrictive and does not apply to most frequent pattern mining problems including the frequent itemset mining problem.
Proof: Let $\mathcal{D}_E$ be a dataset defined over a ground set ground set $E = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ such that $\mathcal{D}_E = [\{A, B\}, \{B, C\}]$. The set of frequent itemset sets $\mathcal{F} = \{\emptyset, \mathrm{A}, \mathrm{B}, \{\mathrm{AB}\}, \{\mathrm{BC}\}\}$ extracted from $\mathcal{D}_E$ with a minimum support value $\varepsilon = 1$ is not confluent. Indeed, $\{\mathrm{AB}\} \cap \{\mathrm{BC}\}$ includes $\{\mathrm{B}\}$ which *is* a pattern in $\mathcal{F}$, yet $\{\mathrm{AB}\} \cup \{\mathrm{BC}\} = \{\mathrm{ABC}\}$ *is not* a pattern in $\mathcal{F}$. Hence $\mathcal{F}$ is not confluent.

In contrast, supported confluence is satisfied for frequent itemsets as it will be shown in Section 3.1.

If the closure is unique, Algorithm 1 is a generic closure operator. In practice, this algorithm may be very costly and can be replaced by more efficient specific algorithms relying on characterizations of the closure operator that exploit particular properties of the problem (see Section 3).

2.3 Formal problem statement

In this paper, we address the problem of closed pattern mining where the closure of every pattern is unique and can be computed with a closure operator.

The general pattern mining problem that we address in this paper can be stated as follow.

Definition 8 (Closed pattern mining problem) Given a ground set E, a dataset $\mathcal{D}_E$, a selection criterion $Select$ and closure operator Clo, extract from $\mathcal{D}_E$ all the closed patterns, that is any set $P \subseteq E$ such that:

1. P occurs in $\mathcal{D}_E$ ($\mathcal{D}_E[P] \neq \emptyset$)
2. $Select(P, \mathcal{D}_E) = true$
3. $Clo(P, \mathcal{D}_E) = P$.

2.4 Enumeration of closed patterns

Generating and testing the complete set of candidate patterns is not feasible in practice because of the exponential number of candidate patterns ($2^{|E|}$). In order to get acceptable performances, closed (and standard) pattern mining algorithms rely on efficient *enumeration strategies*. Enumeration strategies exploit the structured nature of the search space to minimize the number of candidate pattern tests required to find the complete set of closed patterns.

Structured search space: Most pattern mining algorithms perform a constructive enumeration of the closed patterns, using a pattern *augmentation* relation. In our framework, the augmentation relation is defined as follow.

Definition 9 (Pattern augmentation) A pattern Q is an *augmentation* of a pattern P if there exists $e \in Q \setminus P$ such that $Q = P \cup \{e\}$.

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. The Figure 2 (a) is an example of such a DAG structure.

An enumeration strategy is required to explore this DAG and find all the closed patterns without generating duplicates. Arimura and Uno (2009) have shown that it is possible to build polynomial-space enumeration strategies if the *set system* that consists in the ground set and the set of patterns satisfies the property of *accessibility*.

Definition 10 (Set system) A set system is an ordered pair $(E, \mathcal{F})$ where E is a set of elements and $\mathcal{F} \subseteq 2^E$ is a family of subsets of E.

In the context of pattern mining, E is the ground set, and $\mathcal{F}$ is the set of patterns.

Definition 11 (Accessible set system) A set system $(E, \mathcal{F})$ is *accessible* if for every non-empty $X \in \mathcal{F}$, there exists some $e \in X$ such that $X \setminus \{e\} \in \mathcal{F}$.

The intuition behind the notion of accessibility is that when a set system is accessible, there exists in the corresponding DAG, a path connecting $\perp$ to *any other* pattern. This guarantees that it is possible to generate any pattern by augmenting another smaller pattern.

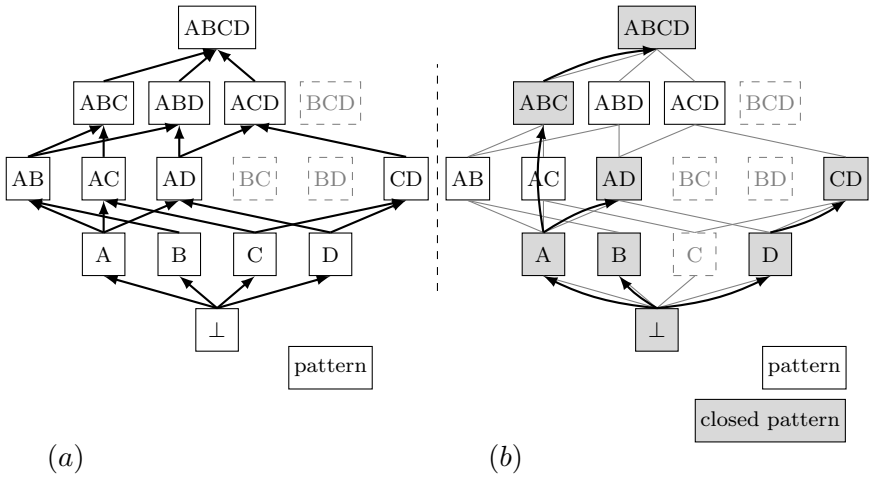

Fig. 2 (a): A DAG representation of set of patterns defined over the ground set $\{A, B, C, D\}$. A solid box labeled ABD represent the pattern $\{A, B, D\}$. Arrows connecting the patterns represent the augmentation relations. (b): The same DAG with closed patterns highlighted (shaded boxes). The thick arrows connecting the closed patterns are the arcs of a possible *enumeration tree*.

Augmenting every pattern will possibly generate duplicate patterns. For example in Figure 2 (a) the augmentation of AD with C and augmentation of AC with D both lead to the pattern ACD. Following an enumeration tree as the one in Figure 2 (b) ensures that no duplicate patterns will be generated. In order to follow an enumeration tree, one must be able to identify a unique parent for every closed pattern. Once a unique parent has been identified for each closed pattern, the strategy is to augment a pattern P into Q only if P is the unique parent of Q. In order to define a unique parent for every closed pattern, we build a lexicographical order over $\mathcal{F}$ (denoted $<_{\mathcal{F}}$) based on a given order over the ground set E. Following Arimura and Uno (2009), we call *first parent* of a pattern, the smallest parent of this pattern with respect to $<_{\mathcal{F}}$.

Definition 12 (First parent) Let P be a closed pattern, and Q the closure of an augmentation $P \cup \{e\}$ of P such that $P <_{\mathcal{F}} P \cup \{e\}$. P is the *first parent* of Q if there does not exist a closed pattern $P' <_{\mathcal{F}} P$ and an element e' such that $P' <_{\mathcal{F}} P' \cup \{e'\}$ and Q is the closure of $P' \cup \{e'\}$.

Designing a tree-based enumeration strategy in accessible set systems boils down to designing a procedure able to detect the *first parent* of every closed pattern. Efficient detection of the first parent is critical to build efficient enumeration strategies.

Storing the set of closed patterns enumerated so far to find the smallest parent pattern (w.r.t. the lexicographical order on $\mathcal{F}$) is not a good strategy. Since the number of closed patterns can grow exponentially with the size of

the ground set it may require exponential memory space. In addition updating the set of closed patterns would required extensive communication and synchronization in the context of a parallel algorithm.

Instead, efficient pattern mining algorithms such as LCM *compute* the first parent without relying on a memory representation of the patterns enumerated so far. This is important because it enables exploration of the search space without global communication. Arimura and Uno (2009) have shown that it is possible to compute the first parent in an accessible set system in polynomial space (thus without retaining patterns in memory). But it requires additional calls to *Select*. This does not take into account the fact that calls to *Select* requires costly accesses to the dataset.

We now present two properties that are stronger than accessibility that allow more efficient computation of the first parent.

Definition 13 (Independence set system) A set system $(E, \mathcal{F})$ is an *independence* set system if $Y \in \mathcal{F}$ and $X \subseteq Y$ together imply $X \in \mathcal{F}$.

Definition 14 (Strongly accessible set system) A set system $(E, \mathcal{F})$ is *strongly accessible* if it is accessible and if for every $X, Y \in \mathcal{F}$ with $X \subset Y$, there exists some $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{F}$.

Any independence set system is strongly accessible (Boley et al, 2010). Any strongly accessible set system is accessible by definition.

When the set system formed by the patterns is an *independence* set system, every subset of a pattern is also a pattern. Therefore checking whether a pattern P is the first parent of Q can be done at almost no cost by checking whether P precedes Q in a lexicographical order over $\mathcal{F}$ based on any order over E.

It is worth noting that this method is correct to mine frequent closed itemsets because the set of frequent itemsets is an independence set system. However many other pattern mining problems such as the graph mining problem that we have introduced at the beginning of this section are not independent. (Proofs are provided in Section 4.)

When the set system formed by patterns is *strongly accessible*, the first parent test is more complex but can be done without performing calls to *Select*. Boley et al (2010) have shown that testing if a closed pattern P is the first parent of a closed pattern Q can be done efficiently by retaining in an *exclusion list* the elements $e \in E$ that are augmentations of a parent of P that have been developed. If a closure Q of P contains any element of E then it will be generated from another branch of the enumeration tree and should not be developed in the current branch.

The principle of closed pattern enumeration in a strongly accessible set system is given in Algorithm 2. This algorithm is a reformulation of Boley et al (2010) algorithm adapted to our framework. The *enum_clo*() procedure builds the augmentations of a pattern P by testing (Line 5) which candidate pattern $P \cup \{e\}$ satisfies the selection criterion *Select*. If $P \cup \{e\}$ satisfies the selection criterion, *enum_clo*() applies the closure operator *Clo* to get the

closure Q of $P \cup \{e\}$. Line 9, we check that P is the *first parent* of Q by testing that Q does not contain any element in EL. If P is the first parent of Q, then $enum\_clo()$ is called with Q as first argument. Then e is added to EL to recall that supersets of $P \cup \{e\}$ must not be explored in next iterations of the *forall* loop of Line 5.

Algorithm 2 Enumerate closed patterns in strongly accessible set systems

Require: An strongly accessible set system $(E, \mathcal{F})$, a closure operator Clo and a dataset $\mathcal{D}_E$.
Ensure: Output the set $\mathcal{C}$ of all the pattern in $\mathcal{F}$ that are closed in $\mathcal{D}_E$.
1: $enum\_clo(Clo(\bot, \mathcal{D}_E), \emptyset)$

2: enum_clo(P, EL
Require: A closed pattern P, and an exclusion list EL.
Ensure: Outputs the set of all the closed patterns that have P as an ancestor in the enumeration tree.
3: *output* P
4: */* Generate all the augmentation of P.*/*
5: **for all** e such that $P \cup \{e\}$ occurs in $\mathcal{D}_E$ **do**
6: **if** $Select(P \cup \{e\}, \mathcal{D}_E)$ **then**
7: $Q \leftarrow Clo(P \cup \{e\}, \mathcal{D}_E)$
8: */* Detect if P is the first parent of Q.*/*
9: **if** $Q \cap EL = \emptyset$ **then**
10: $enum\_clo(Clo(P \cup \{e\}, \mathcal{D}_E), EL)$
11: $EL \leftarrow EL \cup \{e\}$
12: **end if**
13: **end if**
14: **end for**

We illustrate the principle of Algorithm 2 in Figure 3. Figure 3 represents an intermediary step of the process of exploring the enumeration tree in Figure 2 (*b*). When augmenting $\bot$ with element D, $\bot$ has been formerly augmented with A and B, thus EL contains A and B. When trying to augment D with A, we obtain the closed pattern {AD}. However {AD} contains A which is also included in EL thus D is not the first parent of {AD} signaling that {A}D have been enumerated from another branch. (The one starting at A.) We thus consider the next augmentation of D which is {CD}, the closure of

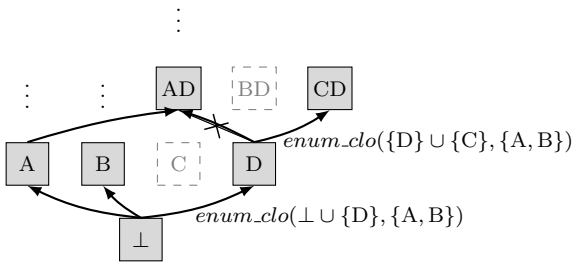

Fig. 3 Enumeration with the exclusion list

{CD} is {CD} it self which does not contain any element in EL. Therefore D is the first parent of {CD}.

In an a strongly accessible set system, it is guaranteed that if {A} and {AD} are two patterns, there exists an augmentation path between {A} and {AD}. (Hidden on Figure 3, visible on Figure 2 (*b*).) Therefore {D} is not the *first* parent of {AD}.

Conversely no subset of {D} could be augmented with C, since {CD} is a pattern, {D} is its first parent.

Two points deserve to be emphasized:

i. The exclusion lists are transmitted from parent to child but not the other way around. Thus branches can be explored independently. Illustrated on Figure 3, it means that is it is not required to explore the branch rooted at {A} to ensure correct exploration of the branch rooted as {D}. In other words, sibling branches can be explored in any order without compromising the soundness or the completeness of the enumeration strategy.
ii. The exclusion list size is $log(|E|)$ thus the space required to compute the first parents is polynomial even if $\mathcal{F}$ is equal to 2^n.

We have shown in this section that mining patterns in independence set systems or in strongly accessible set systems can be done at low cost. However, strongly accessible set systems are more general (they can capture more pattern mining problems) than independence set systems while retaining several important properties such as: polynomial space and output polynomial time complexity with the number of closed pattern, the ability to detect the first parent with local computations and without additional calls to *Select* and accesses to the dataset.

We show in the following Section 3 that strongly accessible set systems are sufficient to capture a broad range of pattern mining problems. We also show that it can be turned into an efficient parallel algorithm that can compete with specialized algorithms.

3 Encoding different pattern mining problem within this generic framework

The framework presented in Section 2 can capture various types of pattern mining problems by using the adequate encoding.

First, representing patterns as sets is powerful enough to mine a broad range of pattern types from *frequent itemsets* to *rigid sequences*, or *picture patterns* (Arimura and Uno, 2009). In this section, we will focus on three distinct pattern mining problems: the problem of mining frequent itemsets, the problem of mining frequent connected relational graphs and the problem of mining gradual itemsets. For each one of them, we will provide a possible encoding into our framework, then we will demonstrate the existence of a closure operator and we will prove the strong accessibility of the corresponding set system.

3.1 Mining closed frequent itemsets (FIM)

Ground set & dataset: Encoding a frequent itemset input dataset is direct and has been explained in Section 2.1.

Selection criterion: A subset of the ground set E is a pattern if it occurs in at least ε transactions (for a given constant ε). For any $P \subseteq E$, $Select(P, \mathcal{D}_E) \equiv |\mathcal{D}_E[P]| \geq \varepsilon$.

Closure Uniqueness: The closure uniqueness for the FIM problem is a consequence of the fact that a set of frequent itemsets is support confluent.

Theorem 2 (Supported confluence for the FIM problem) *The FIM problem is support confluent.*

Proof: We recall that $\mathcal{F}$ is support confluent with respect to a dataset $\mathcal{D}_E$ if for every two patterns Q and Q' in $\mathcal{F}$ such that $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$, $\exists Z \subseteq Q \cap Q' \neq \emptyset$ implies that $Q \cup Q'$ belongs to $\mathcal{F}$. Let Q and Q' be two candidate patterns such that $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$. The support set of $Q \cup Q'$ is the set of transactions containing both Q and Q': $\mathcal{D}_E[Q \cup Q'] = \mathcal{D}_E[Q] \cap \mathcal{D}_E[Q']$. Since $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$ by hypothesis, $\mathcal{D}_E[Q] \cap \mathcal{D}_E[Q'] = \mathcal{D}_E[Q]$. Since Q is frequent, $|\mathcal{D}_E[Q]| \geq \varepsilon$ hence $|\mathcal{D}_E[Q \cup Q'] \geq \varepsilon$. Thus if Q is a pattern $Q \cup Q'$ is also a pattern. □

Strong Accessibility: The set system $(E, \mathcal{F})$ formed by the frequent itemsets have been proved to be an independence set system by Boley et al (2010).

3.2 Mining closed frequent connected relational graphs (CRG)

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). An example of a relation graph dataset has been presented in Figure 1.

The problem of mining frequent connected relation graph can be stated as follows: Given a set of vertices V and a set of relational graphs $G_1, \ldots, G_n$ where each G_i is a relational graph (V, E_i) defined using the nodes in V, extract the connected sub-graphs occurring in at least ε input graphs.

The problem of extracting frequent closed connected relational graphs can be captured in our setting as follows:

Ground set: The ground set E is a set of pairs in $V \times V$, each pair is used to represent an edge connecting two nodes in V.

Dataset: The dataset $\mathcal{D}_E = [t_1, \ldots, t_n]$ is a sequence of transactions where for all $i \in [1, n]$, the transaction $\mathcal{D}_E(t_i)$, represents the input graph G_i. Each element in the transaction t_i is a pair representing an edge in G_i.

Selection criterion: Given a candidate pattern G, $Select(G, \mathcal{D}_E)$ returns true if and only if:

- G is a connected set of edges.
- $|\mathcal{D}_{E_g}[G]| \geq \varepsilon$ (for a given constant ε)

Closure uniqueness: The closure for the CRG problem is unique because it is support confluent.

Theorem 3 (Supported confluence for the FIM problem) *The CRG problem is support confluent.*

Proof: Let Q and Q' be two patterns in $\mathcal{F}$ such that $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$. and let Z be a pattern in $\mathcal{F}$ such that $Z \subseteq Q \cap Q'$ and $Z \neq \emptyset$. As patterns, Q, Q' and Z are connected set of edges. Q is connected implies that for every edges $e, e' \in Q$, there exists a path $[e, \ldots, e']$, connecting e and e'. Let e, be an edge in $Q \cap Q'$ (This edge always exists because $Q \cap Q' \supseteq Z \neq \emptyset$) then $\forall e' \in Q$ and $e'' \in Q'$, there exists a path $[e', \ldots, e, \ldots, e''] \in Q \cup Q'$. Hence $Q \cup Q'$ is connected. In addition since $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$, every transaction in $\mathcal{D}_E[Q]$ also contains Q' hence $|\mathcal{D}_E[Q \cup Q']| = |\mathcal{D}_E[Q]|$. By hypothesis $|\mathcal{D}_E[Q]| \geq \varepsilon$ hence, $|\mathcal{D}_E[Q \cup Q']| \geq \varepsilon$ and $Q \cup Q'$ is also frequent. Since $Q \cup Q'$ is frequent and connected, it is a pattern. $\mathcal{F}$ is support confluent. □

Closure operator: The closure of a graph P is the set of edges connected to P occurring in every transactions of the support set of P. It can be computed with Algorithm 3. This algorithm is a specialization of Algorithm 1: in Line 3, we only perform a connectivity test rather than a complete call to *Select* because by construction $Q \cup \{e\}$ is frequent. (Q is a frequent pattern, and $\{e\}$ has the same support set than Q, therefore $Q \cup \{e\}$ is frequent.)

Algorithm 3 Closure operator for the CRG problem

Require: a graph pattern P and a dataset $\mathcal{D}_E$
Ensure: returns the unique closure Q of P.
1: $Q \leftarrow P$
2: */*while there exists e such that e is connected to Q*/*
3: **while** $\exists e \in \cap \mathcal{D}_E[P] \setminus Q$ such that Q is connected e **do**
4: $\quad Q \leftarrow Q \cup \{e\}$
5: **end while**
6:
7: **return** Q

Strong accessibility: Theorem 5 states that the set system corresponding to the CRG problem is strongly accessible.

However, in order to prove Theorem 5 we need Theorem 4 that shows that the intersection of an independent set system and a strongly accessible set system is strongly accessible.

Theorem 4 (Set system intersection) *Given two set systems $S_1 = (E, \mathcal{F}_1)$ and $S_2 = (E, \mathcal{F}_2)$ defined over the same ground set E, if S_1 is independent and S_2 is strongly accessible, the set system $S_3 = (E, \mathcal{F}_1 \cap \mathcal{F}_2)$ is then strongly accessible.*

Proof: We denote F_3 the intersection of $\mathcal{F}_1$ and $\mathcal{F}_2$, hence $\mathcal{S}_3 = (E, \mathcal{F}_3)$. Let Y be a subset of E in $\mathcal{F}_3$, and let X be any subset of Y. Since $\mathcal{F}_3 = \mathcal{F}_1 \cap \mathcal{F}_2$, Y is also in $\mathcal{F}_1$ and $\mathcal{F}_2$. As a subset of Y, X is also in $\mathcal{F}_1$, therefore any subset X of Y belongs to $\mathcal{F}_3$ if and only if it belongs to $\mathcal{F}_2$.

We now show that S_3 is accessible. $\mathcal{F}_2$ is strongly accessible thus accessible, therefore there exists $e \in Y$ such that $Y \setminus \{e\} \in \mathcal{F}$. However $Y \setminus \{e\}$ is a subset of Y and belongs to $\mathcal{F}_2$, therefore it also belongs to $\mathcal{F}_3$. S_3 is accessible.

In a similar way, we show that S_3 is also strongly accessible. Since $\mathcal{F}_2$ is strongly accessible there exists some $e \in X \setminus Y$ such that $X \cup \{e\} \in \mathcal{F}_2$. However, $X \cup \{e\}$ is a subset of Y and therefore also belongs to $\mathcal{F}_3$. S_3 is accessible and for any $Y, X \in \mathcal{F}_3$, there exists $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{F}_3$ therefore S_3 is strongly accessible. □

Theorem 5 (Strong accessibility for CRG) *The set system associated with the CRG problem is strongly accessible for every dataset $\mathcal{D}_E$.*

Proof: We denote $(E, \mathcal{F}_0)$, with $\mathcal{F}_0 = Select(2^E, \mathcal{D}_E)$ the set system formed by all the sets $X \subseteq E$ satisfying the selection criterion. That is, any X such that:

- X is a frequent set of edges in $\mathcal{D}_E$,
- X is a connected set of edges.

Let $\mathcal{F}_1$ be the set of sets $X \subseteq E$ such that X is a frequent set of edges and let $\mathcal{F}_2$ the set of sets $Y \subseteq E$ such that Y is a connected set of edges. Hence, $\mathcal{F} = \mathcal{F}_1 \cap \mathcal{F}_2$ is set of all the frequent and connected sets of edges $\mathcal{F}_0 = \mathcal{F}_1 \cap \mathcal{F}_2$.

We observe that mining frequent sets of edges is equivalent to mine frequent set of items, hence the set system $(E, \mathcal{F}_1)$ formed by all the frequent set of edges is an independence set system.

The set system $(E, \mathcal{F}_2)$ formed by all the connected sets of edges have been proved to be strongly accessible in (Boley et al, 2010).

According to Theorem 4, the set system formed by all the frequent and connected graphs $(E, \mathcal{F}_0 = \mathcal{F}_1 \cap \mathcal{F}_2)$ is strongly accessible. □

3.3 Mining closed gradual itemsets (GRI)

The problem of mining gradual itemsets consists in mining attribute co-variations in numerical databases (Ayouni et al, 2010). Consider the numerical database in Table 1.

When considering the records p_1, p_2 and p_3, it appears that an increase in temperature is correlated with a decrease in electric consumption. This co-variation of the temperature and the electric consumption can be represented

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 of a numerical database

by the *gradual itemset* $(T^{\uparrow}, EC^{\downarrow})$ (where T stands for *Temperature* and EC stands for *Electric Consumption*). This gradual itemset is *respected* by the sequence of records $[p_1, p_2, p_3]$. Note that symmetrically, $(T^{\downarrow}, EC^{\uparrow})$ is respected by $[p_3, p_2, p_1]$.

Let $A = \{a_1, \ldots, a_m\}$ be a set of attributes and $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of records where each record p_i with $i \in [1, n]$ stores a numerical value for every attribute in A. The problem of mining closed gradual itemsets can be represented in our framework by considering as ground set the variations of attributes and by encoding transactions and patterns as subsets of attribute variations verifying some constraints. The encoding is the following:

Ground set: E is the set of attributes variations: $E = \{a_1^{\uparrow}, a_1^{\downarrow}, \ldots, a_m^{\uparrow}, a_m^{\downarrow}\}$.

Dataset: In the dataset $\mathcal{D}_E$, there are as many transactions as pairs of records $(p_i, p_j) \in \mathcal{P}$ with $i, j \in [1, n]$ and $i \neq j$. A transaction has as identifier (p_i, p_j) if it contains the variation for every attribute in A between the records p_i and p_j. We will denote the corresponding transaction $t_{(p_i,p_j)}$: for every attribute $a \in A$, $a^{\uparrow} \in t_{(p_i,p_j)} \Leftrightarrow p_i[a] \leq p_j[a]$ ($p[a]$ denoting the value of attribute a for record p), $a^{\downarrow} \in t_{(p_i,p_j)}$ otherwise. The corresponding encoded dataset for the database in Table 1 is shown in Table 2.

$$
\begin{array}{ll}
t_{(p_1,p_2)}: & [\{T^{\uparrow}, EC^{\downarrow}\},\\
t_{(p_1,p_3)}: & \{T^{\uparrow}, EC^{\downarrow}\},\\
t_{(p_1,p_4)}: & \{T^{\uparrow}, EC^{\downarrow}\},\\
t_{(p_2,p_1)}: & \{T^{\downarrow}, EC^{\uparrow}\},\\
t_{(p_2,p_3)}: & \{T^{\uparrow}, EC^{\downarrow}\},\\
t_{(p_2,p_4)}: & \{T^{\uparrow}, EC^{\uparrow}\},\\
t_{(p_3,p_1)}: & \{T^{\downarrow}, EC^{\uparrow}\},\\
t_{(p_3,p_2)}: & \{T^{\downarrow}, EC^{\uparrow}\},\\
t_{(p_3,p_4)}: & \{T^{\uparrow}, EC^{\uparrow}\},\\
t_{(p_4,p_1)}: & \{T^{\downarrow}, EC^{\uparrow}\},\\
t_{(p_4,p_2)}: & \{T^{\downarrow}, EC^{\downarrow}\},\\
t_{(p_4,p_3)}: & \{T^{\downarrow}, EC^{\downarrow}\}]
\end{array}
$$

Table 2 Encoding of the database in Table 1: each transaction contains the variation of the attributes T and EC between each record in the database. A variation can be positive ($\uparrow$) or negative ($\downarrow$).

Selection criterion: Given a constant ε, and a candidate pattern $G = \{a_{g_1}^{v_1}, \ldots, a_{g_k}^{v_k}\}$ with $g_1 < \ldots < g_k$, and $v_1, \ldots, v_k$ variations of the form $\uparrow$ or $\downarrow$, G is a pattern if the size of the longest *tid path* in the tid support set of G is greater or equal than ε.

Definition 15 (tid path) A tid path is a sequence of transactions identifiers $[(p_{i_1}, p_{j_1}), \ldots, (p_{i_n}, p_{j_n})]$ such that $\forall k \in [1, n[,\ p_{j_k} = p_{i_{k+1}}$.

For example the longest tid path in $\{(p_1, p_2), (p_1, p_3), (p_1, p_4), (p_2, p_3)\}$ is $[(p_1, p_2), (p_2, p_3)]$ of size 2.

When it is clear from context, we refer to the longest tid path in the tid support set of G as the longuest tid path of G.

In addition one can observe that if a pattern is frequent, then a symmetric pattern is also frequent and does not carry additional information. For example: $(T^{\uparrow}, EC^{\downarrow})$ and $(T^{\downarrow}, EC^{\uparrow})$. To take into account the symmetry of the problem we discard from the set of patterns, any pattern $G = \{a_{g_1}^{v_1}, \ldots, a_{g_k}^{v_k}\}$ such that v_1 is not $\uparrow$.

Thus, given a pattern $G = \{a_{g_1}^{v_1}, \ldots, a_{g_k}^{v_k}\}$, $Select(G, \mathcal{D}_E)$ returns true if and only if:

1. The size of the longest tid path in the tid support set of G is greater or equal to ε,
2. G is empty or the first variation v_1 of a_{g_1} in G is $\uparrow$.

Closure uniqueness: The closure for the GRI problem is unique because it is support confluent.

Theorem 6 (Supported confluence for the GRI problem) *The GRI problem is support confluent.*

Proof: Let Q and Q' be two patterns in $\mathcal{F}$ such that $\mathcal{D}_E[Q] = \mathcal{D}_E[Q']$ and Z be another pattern in $\mathcal{F}$ such that $Z \subseteq Q \cap Q'$ and $Z \neq \emptyset$. Q is a pattern, therefore there exists in $\mathcal{D}_E[\![Q]\!]$ a tid path of size greater or equal than ε. Since Q and Q' have the same support set in $\mathcal{D}_E$, $\mathcal{D}_E[\![Q]\!] = \mathcal{D}_E[\![Q']\!] = \mathcal{D}_E[\![Q \cup Q']\!]$ and the same tid path also exist in $\mathcal{D}_E[\![Q \cup Q']\!]$. Therefore $Q \cup Q'$ is pattern if and only if the first variation v_1 of $a_{g_1}^{v_1}$ is $\uparrow$, which is always granted since Q and Q''s first variations are both $\uparrow$. Hence $Q \cup Q' \in \mathcal{F}$. $\mathcal{F}$ is support confluent. □

Closure operator: We recall that the attributes $a_1, \ldots, a_m$ are ordered. The closure of a pattern P is the intersection of the transactions in $\mathcal{D}_E[P]$, from which we remove the descending variations of the attributes before the first attribute with an ascending variation. Algorithm 4 is more efficient than the generic closure operator given in Algorithm 1. Theorem 7 states that both algorithms are equivalent in the context of the GRI problem.

Note that the closure operator proposed in 4 is defined in a much simpler way than the one proposed by Di-Jorio et al (2009). This simplification is due to the set-based encoding of the problem.

Algorithm 4 Closure operator for the GRI problem encoding attribute variations

Require: a gradual itemset pattern P and a dataset $\mathcal{D}_E$ encoding attribute variations
Ensure: returns the unique closure Q of P.
1: $Q_{max} \leftarrow \cap \mathcal{D}_E[P]$
2: $a_1 \leftarrow$ the first attribute with an ascending variation in Q_{max}
3: $Q \leftarrow Q_{max}$
4: **while** $\exists a^{\downarrow} \in Q$ such that a precedes a_1 in the order of the attributes **do**
5: $\quad Q \leftarrow Q \setminus \{a^{\downarrow}\}$
6: **end while**
7:
8: **return** Q

Theorem 7 (Closure operator for the GRI problem) *Algorithm 4 computes the unique closure of every gradual itemset.*

Proof: We show that the closure operator defined in Algorithm 4 for the GRI problem is equivalent to the generic closure operator defined in Algorithm 1.

Let a_1 be the first attribute with an ascending variation in $\bigcap \mathcal{D}_E[P]$.

First, let us show that for any $a^{\downarrow} \in \bigcap \mathcal{D}_E[P]$ such that a is before a_1 in the order of the attributes, there does not exist $S \subset \bigcap \mathcal{D}_E[P]$ such that $a^{\downarrow} \in S$ and $P \cup S$ is a pattern.

Suppose that $P \cup S$ is a pattern: its first variation is ascending and therefore there is $b^{\uparrow}$, where $b < a < a_1$. This contradicts the fact that a_1 is the first attribute with an ascending variation in $\bigcap \mathcal{D}_E[P]$. Therefore the closure of P cannot include any element in the set R of elements suppressed in Line 5 of Algorithm 4.

Second, let us show that $S' = \bigcap \mathcal{D}_E[P] \setminus R$ is the closure of P. P being a pattern, $\mathcal{D}_E[\![P]\!]$ contains a tid path of size ε. By construction S' has the same support set as P therefore the tid support set $\mathcal{D}_E[\![S']\!]$ of S' also contain the same tid path.

By removing R from $\bigcap \mathcal{D}_E[P]$ to build S', it is guaranteed that there is no descending variation on attributes before a_1 in S'. Therefore S' is a pattern.

By definition elements that are not in $\bigcap \mathcal{D}_E[P]$, cannot be in the closure of P, hence S' is maximal and thus is the closure of P. □

Strong accessibility: We state in Theorem 8 that the set system corresponding to the GRI problem is strongly accessible.

Theorem 8 (Strong accessibility for GRI) *The set system associated with the GRI problem is strongly accessible for every dataset $\mathcal{D}_E$.*

Proof: We denote $(E, \mathcal{F}_0)$, with $\mathcal{F}_0 = Select(2^E, \mathcal{D}_E)$ the set system formed by all the set $X \subseteq E$ satisfying the selection criterion. That is, any $X = \{a_{x_1}^{v_1}, \ldots, a_{x_k}^{v_k}\}$ such that:

- X the longuest path in the tid support set of X is greater or equal than ε. $(P1)$

– X is empty or, the first variation v_1 of a_{x_1} in X is $\uparrow$. $(P2)$

Let $\mathcal{F}_1$ be the set of sets $X \subseteq E$ satisfying $(P1)$ and, $\mathcal{F}_2$ the set of sets $Y \subseteq E$ satisfying $(P2)$. The set of sets satisfying both conditions is $\mathcal{F}_0 = \mathcal{F}_1 \cap \mathcal{F}_2$.

We show that $(E, \mathcal{F}_1)$ is an independence set system and that $(E, \mathcal{F}_2)$ is a strongly accessible set system. Hence we can apply Theorem 4.

$(E, \mathcal{F}_1)$ **is an independence set system:** $\forall Y \in \mathcal{F}_1$ with $X \subseteq Y$, $X \in \mathcal{F}_1$ if and only if the tid support set of Y in $\mathcal{D}_E$ contains a tid path of size greater than ε. Since $X \subseteq Y$, $\mathcal{D}_E[Y] \subseteq \mathcal{D}_E[X]$ and $\mathcal{D}_E[\![Y]\!] \subseteq \mathcal{D}_E[\![X]\!]$, therefore the same tid path also exists in $\mathcal{D}_E[\![X]\!]$. Thus $X \in \mathcal{F}_1$. the set system F_1 an independence set system.

$(E, \mathcal{F}_2)$ **is strongly accessible:** We recall that $(E, \mathcal{F}_2)$ is strongly accessible if and only if:

1. it is accessible
2. for every $X, Y \in \mathcal{F}$ with $X \subset Y$, there exists some $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{F}$.

We first prove that the set system $(E, \mathcal{F}_2)$ is accessible. From $(P2)$, $Y = \{a_{y_1}^{v_1}, \ldots, a_{y_k}^{v_k}\} \in \mathcal{F}_2$ implies that the variation v_1 of the first attribute a_{y_1} in Y, is $\uparrow$. We show that there always exists $a_{y_i}^{v_i} \in Y$, with $i \in [1, k]$, such that $Y \setminus \{a_{y_i}^{v_i}\}$ still satisfies $(P2)$. For every $Y \in \mathcal{F}_2$, if $|Y| \geq 2$, there exists at least one $a_{y_i}^{v_i} \in Y$ with $i \neq 1$. Hence there exists $a_{y_i}^{v_i}$, such that $X = Y \setminus \{a_{y_i}^{v_i}\}$ contains $a_{y_1}^{v_1}$. Since $a_{y_1}^{v_1}$, is the variation for the first attribute in X, and $v_1 = \uparrow$, $X = Y \setminus \{a_{y_i}^{v_i}\}$ satisfies $(P2)$. In addition, if $|Y| = 1$, $X = Y \setminus \{a_{y_1}^{v_1}\} = \emptyset$ for every $e \in Y$, which is granted to be in $\mathcal{F}_2$ since the restriction $(P2)$ does not apply to $\emptyset$. Therefore for every $Y \in \mathcal{F}_2$ there exists at least one $a_{y_i}^{v_i}$ such that $Y \setminus \{a_{y_i}^{v_i}\} \in \mathcal{F}_2$. (E, F_2) is accessible. The set system $(E, \mathcal{F}_2)$ is accessible.

We show that for every $X, Y \in \mathcal{F}$ with $X \subset Y$, there exists some $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{F}$. Let $X = \{a_{x_1}^{v_1}, \ldots, a_{x_k}^{v_k}\}$ and $Y = \{a_{y_1}^{v_1}, \ldots, a_{y_p}^{v_p}\}$ be two patterns in $\mathcal{F}_2$ with $X \subset Y$, since (E, F_2) is accessible, it also is strongly accessible if and only if $\forall X, Y \in \mathcal{F}_2$ with $X \subset Y$, there exists $a_{y_i}^{v_i} \in Y \setminus X$, such that $X \cup \{a_{y_i}^{v_i}\} \in \mathcal{F}_2$.

– If $|Y| - |X| = 1$ then there exists only one $a_{x_i}^{v_i} \in Y \setminus X$ and $Y \setminus \{a_{x_i}^{v_i}\} = X$. Since $X \in \mathcal{F}_2$, $Y \setminus \{a_{x_i}^{v_i}\} = X$ is in $\mathcal{F}_2$ as well.
– If $|Y| - |X| \geq 2$, let $a_{y_i}^{v_i}$ be any attribute variation in $Y \setminus X$ with $a_{y_i}^{v_i} \neq a_{y_1}^{v_1}$. Then $Y \setminus \{a_{y_i}^{v_i}\}$ admits $a_{y_1}^{v_1}$ as a first variation with $v_1 = \uparrow$. $Y \setminus \{a_{y_i}^{v_i}\} \in \mathcal{F}_2$.

$(E, \mathcal{F}_2)$ is a strongly accessible.

From Theorem 4, if $(E, \mathcal{F}_1)$ is an independence set system, and $(E, \mathcal{F}_2)$ is a strongly accessible set system, the set system $(E, \mathcal{F}_0 = \mathcal{F}_1 \cap \mathcal{F}_2)$ is strongly accessible. □

4 The ParaMiner algorithm

In this section, we present ParaMiner, a parallel algorithm able to solve pattern mining problems formalized in the framework described in Section 2. ParaMiner relies on the enumeration strategy given in Algorithm 2.

ParaMiner is an extension of Algorithm 2 in which we have introduced a novel dataset reduction technique called *EL-reduction*. *EL-reduction* can be used for pattern mining problems expressed in our framework and enables efficient pattern testing that is required to build efficient pattern mining algorithms.

The soundness of the *EL-reduction* relies on the *decomposability* property that we will introduce in Section 4.3. The soundness and the completeness of ParaMiner are then proved for pattern mining problem whose selection criterion are decomposable.

4.1 ParaMiner: main algorithm

The main procedure of ParaMiner is given in Algorithm 5. This procedure computes the closure of $\bot$ and calls the recursive procedure *expand*() (Algorithm 6) to compute and output the remaining closed patterns.

Algorithm 5 ParaMiner

Require: ground set E, selection criterion $Select$, closure operator Clo, dataset $\mathcal{D}_E$
Ensure: Outputs all closed patterns occurring in $\mathcal{D}_E$.
1: **output** $Clo(\bot, \mathcal{D}_E)$
2: $expand(Clo(\bot, \mathcal{D}_E), \mathcal{D}_E, \emptyset)$

Algorithm 6 Expanding a closed pattern P

1: $expand(P, \mathcal{D}_P^{reduced}, EL)$
Require: A closed pattern P, a reduced dataset $\mathcal{D}_P^{reduced}$, an exclusion list EL.
Ensure: Output all closed patterns that are descendent of P in the enumeration tree.
2: **for all** e such that e occurs in $\mathcal{D}_P^{reduced}$ **do**
3: **if** $Select(P \cup \{e\}, \mathcal{D}_P^{reduced})$ **then**
4: $Q \leftarrow Clo(P \cup \{e\}, \mathcal{D}_P^{reduced})$
5: **if** $EL \cap Q = \emptyset$ **then**
6: /* *P is Q's first parent.* */
7: **output** Q
8: $\mathcal{D}_Q^{reduced} \leftarrow reduce(\mathcal{D}_P^{reduced}, e, EL)$
9: **spawn** $expand(Q, \mathcal{D}_Q^{reduced}, EL)$
10: $EL \leftarrow EL \cup \{e\}$
11: **end if**
12: **end if**
13: **end for**

Given a closed pattern P, an individual call to *expand*() is in charge of outputting every closed pattern that is a child of P in the enumeration tree. When *expand*() finds a closed pattern Q that is a descendant of P, it calls the *reduce*() function that produces the reduced dataset $\mathcal{D}_Q^{reduced}$ of the closed pattern Q. The *expand*() procedure is then called again to build the descendants of Q using the reduced dataset $\mathcal{D}_Q^{reduced}$ as illustrated in Figure 4.

In Line 9 instead of performing a sequential call *expand*(), we *spawn* a new task that can be performed on a different processor.

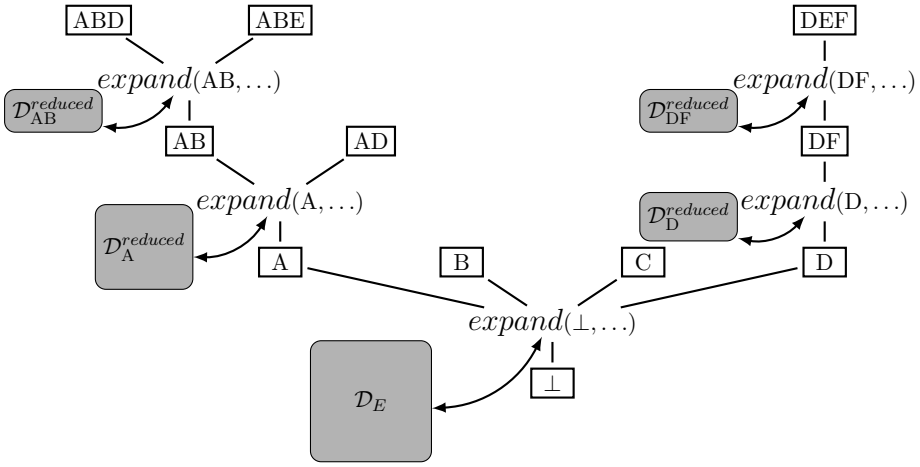

Fig. 4 The *expand*() procedure explores the search space following the enumeration tree (white boxes and lines). In each call *expand*() the augmentations of a closed pattern are built using the reduced dataset of this pattern (shaded boxes).

4.2 Dataset reduction: principles and properties

Dataset reduction techniques were introduced by Han et al (2000) and later improved by Uno et al (2004). These techniques are based on the observation that most patterns occur in a small part of the dataset, hence the whole dataset is not required to compute the selection criterion for a given set of candidate patterns. However these techniques were developed in the context of specific algorithms for frequent itemset mining. We further extend this technique and make it correct for pattern mining problems that have a corresponding set system that is strongly accessible.

The practical efficiency of ParaMiner relies on the fact that the full dataset is replaced by appropriate reduced datasets in the computation of *Clo* (Line 4 of Algorithm 6), *Select* (Line 3 of Algorithm 6) and in the elements chosen to expand closed patterns (Line 2 of Algorithm 6). Reduced datasets are produced by the *reduce*() function presented in Algorithm 7.

In the following, we denote e the element used to obtain Q by augmenting P: $Q = Clo(P \cup \{e\})$.

Removing transactions: Transactions are removed Line 2 of Algorithm 7 when initializing $\mathcal{D}^{reduced}_{Q}$ with the support set of $\{e\}$ in $\mathcal{D}^{reduced}_{P}$. This first reduction step has been introduced by FP-Growth (Han et al, 2000) and consists in removing any transaction not including Q from the dataset.

Algorithm 7 The dataset reduction algorithm

1: $reduce(\mathcal{D}_P^{reduced}, e, EL)$
Require: The reduced dataset $\mathcal{D}_P^{reduced}$ of the parent P of Q, the augmenting element e such that $Q = Clo(P \cup \{e\}, \mathcal{D}_E)$, the exclusion list EL.
Ensure: Returns the reduced dataset of Q: $\mathcal{D}_Q^{reduced}$
2: $\mathcal{D}_Q^{reduced} \leftarrow \mathcal{D}_P^{reduced}[\{e\}]$
3: */* Suppress elements from transactions. */*
4: **for all** $G \in partition(\mathcal{D}_Q^{reduced}, EL)$ **do**
5: **for all** $e \in EL$ **do**
6: **if** there exists $t' \in G$ such that $e \notin t'$ **then**
7: Suppress e from all the transactions in G
8: **end if**
9: **end for**
10: **end for**
11:
12: **return** $\mathcal{D}_Q^{reduced}$

Suppressing elements from transactions: By construction, elements in the exclusion list EL cannot appear in patterns enumerated from Q. These elements however cannot be directly suppressed from $\mathcal{D}_Q^{reduced}$, because some of them are needed to compute the closure of patterns enumerated from Q (Line 4 of Algorithm 6) and check that these patterns have an empty intersection with EL (Line 5). The elements of the exclusion list EL that can be safely removed from transactions in $\mathcal{D}_Q^{reduced}$ are the ones that are guaranteed not to appear in any closed pattern that is a descendent of Q in the enumeration tree.

For example, given the ground set $E = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$, let $\{\mathrm{C}\}$ be a closed pattern, and $EL = \{\mathrm{A}, \mathrm{B}\}$ an exclusion list. Considering the following dataset (each line is a transaction):

$$\mathcal{D}_{\{\mathrm{C}\}}^{reduced} = \begin{array}{|cccc|}\hline & & \downarrow & \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} \\ \mathrm{A} & & \mathrm{C} & \mathrm{D} \\ & & \mathrm{C} & \\ \hline\end{array}$$

In this dataset, the only superset of $\{\mathrm{C}\}$ that can be a closed pattern and a descendent of $\{\mathrm{C}\}$ in the enumeration tree is the candidate pattern $\{\mathrm{C}, \mathrm{D}\}$. Any other superset of Q would necessarily include A or B which belong to the exclusion list, hence it would fail the first parent test (Line 5 in Algorithm 6).

However A must be kept in $\mathcal{D}_Q^{reduced}$ because it is a possible element of the closure of $\{\mathrm{C}, \mathrm{D}\}$. Indeed $\{\mathrm{C}, \mathrm{D}\}$ and $\{\mathrm{A}\}$ have the same support set, hence if $Select(\{\mathrm{A}, \mathrm{C}, \mathrm{D}\}) = true$, then the closure of $\{\mathrm{C}, \mathrm{D}\}$ will be $\{\mathrm{A}, \mathrm{C}, \mathrm{D}\}$. Without A in the dataset the closure of $\{\mathrm{C}, \mathrm{D}\}$ cannot be computed correctly and the first parent test is not guaranteed to fail as it should be. This scenario could lead to the generation of $\{\mathrm{C}, \mathrm{D}\}$ as a closed pattern even if it is not the case.

B does not have the same support set as any superset of $\{\mathrm{C}\}$ that is a descendent of $\{\mathrm{C}\}$ in the enumeration tree, therefore it can be removed from $\mathcal{D}_Q^{reduced}$ without altering neither the soundness nor the completeness of ParaMiner.

To easily determine the elements that cannot belong to any closure of a descendent of Q, $\mathcal{D}_Q^{reduced}$ is partitioned (by the function $partition(\mathcal{D}_Q^{reduced}, EL)$ called Line 4 in Algorithm 7) in groups of sets of transactions that have the same elements except elements of EL (the *partition*() function is detailed in Algorithm 8). 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. (These elements are suppressed Lines 5–9 of Algorithm 7.)

Algorithm 8 The partition function used in *reduce*()

```
 1: partition(D_Q^reduced, EL)
Require: A dataset D_Q^reduced, an exclusion list EL
Ensure: Returns sets of transactions that are equal when considering only the elements
    that are not in EL.
 2: T ← D_Q^reduced
 3: for all t ∈ T do
 4:    G ← {t}
 5:    T ← T \ {t}
 6:    for all t' ∈ T do
 7:       if t \ EL = t' \ EL then
 8:          /* t and t' have the same set of non EL-elements. */
 9:          /* They belong to the same group of transactions. */
10:          G ← G ∪ {t'}
11:          T ← T \ {t'}
12:       end if
13:    end for
14:    G ← G ∪ {G}
15: end for
16:
17: return G
```

The following theorem characterizes the dataset reduction. It states that the transaction identifiers (*tid support sets*, see Definition 4) are preserved by dataset reduction, and the elements that belong to *all* the transactions in the support set of a pattern augmentation are also preserved.

Theorem 9 *Let* $Q = Clo(P \cup \{e\}, \mathcal{D}_P^{reduced})$ *and* $\mathcal{D}_Q^{reduced}$ *be the result of* $reduce(\mathcal{D}_P^{reduced}, e, EL)$. *If* e' *occurs in* $\mathcal{D}_Q^{reduced}$ *and* $(Q \cup \{e'\}) \cap EL = \emptyset$ *then:*

- $\mathcal{D}_E[\![Q \cup \{e'\}]\!] = \mathcal{D}_Q^{reduced}[\![\{e'\}]\!]$.
- $\bigcap \mathcal{D}_E[Q \cup \{e'\}] = \bigcap \mathcal{D}_Q^{reduced}[\{e'\}]$.

Proof: Let us prove the first item. Let i be a tid in $\mathcal{D}_E[\![Q \cup \{e'\}]\!]$: it is the identifier of a transaction t in $\mathcal{D}_E$ including $Q \cup \{e'\}$. t cannot have been removed by Line 2 of *reduce*() because the support set of $\mathcal{D}_E[Q \cup \{e'\}]$ is included in $\mathcal{D}_E[Q]$. In addition the elements in $Q \cup \{e'\}$ cannot have been removed from t

by Line 7 because $(Q \cup \{e'\}) \cap EL = \emptyset$. Therefore the resulting transaction t' obtained by removing elements from t necessarily is in $\mathcal{D}_Q^{reduced}[e']$. Therefore i is also in $\mathcal{D}_Q^{reduced}[\![e']\!]$.

Conversely, since $e' \notin EL$, e' occurs in $\mathcal{D}_Q^{reduced}$ and thus there exists a tid i in $\mathcal{D}_Q^{reduced}[\![e']\!]$. Let t' the transaction identified by i in $\mathcal{D}_Q^{reduced}[\![e']\!]$ and let t the transaction identified by i in $\mathcal{D}_E$. By construction: $Q \cup \{e'\} \subseteq t'$ and $t' \subseteq t$, and thus $Q \cup \{e'\} \subseteq t$. Therefore i is in $\mathcal{D}_E[\![Q \cup \{e'\}]\!]$.

Let us now prove the second item. Let a in $\bigcap \mathcal{D}_E[Q \cup \{e'\}]$. By construction each group of transactions in $\mathcal{D}_E$ including $Q \cup \{e'\}$ also includes a. Hence the condition in Line 6 in Algorithm 7 is not satisfied for a. Therefore a is not suppressed by *reduce*() from any transaction in $\mathcal{D}_Q^{reduced}[\{e'\}]$ and thus it belongs to $\bigcap \mathcal{D}_Q^{reduced}[\{e'\}]$.

Since $e' \notin EL$, e' occurs in $\mathcal{D}_Q^{reduced}$ and thus there exists a in $\bigcap \mathcal{D}_Q^{reduced}[\{e'\}]$. Suppose that a is not in $\bigcap \mathcal{D}_E[Q \cup \{e'\}]$: there exists a transaction t in $\mathcal{D}_E[Q \cup \{e'\}]$ including $Q \cup \{e'\}$ that does not include a. This transaction cannot have been removed by Line 2 of *reduce*() because the support set of $\mathcal{D}_E[Q \cup \{e'\}]$ is included in $\mathcal{D}_E[Q]$. In addition the elements in $Q \cup \{e'\}$ cannot have been removed from t by Line 7 because $(Q \cup \{e'\}) \not\subseteq EL$. Therefore the resulting transaction t' obtained by removing elements from t necessarily is in $\mathcal{D}_Q^{reduced}[\{e'\}]$. Since a is not in t it cannot be in t' either. This contradicts the fact a is in $\bigcap \mathcal{D}_Q^{reduced}[\{e'\}]$. □

4.3 ParaMiner: soundness and completeness

We have shown in Theorem 9 that reductions preserve the integrity of closed patterns and their tid support set. We now introduce the decomposability property which is sufficient to guarantee that the selection criterion can be correctly computed in a reduced dataset. A selection criterion $Select(P, \mathcal{D}_E)$ is decomposable if it can be expressed as the conjunction of two predicates: one over the pattern P and one over the *tid support set* of P in $\mathcal{D}_E$.

Definition 16 (Decomposability of *Select*) Given a ground set E, the *Select* predicate is decomposable if and only if for every dataset $\mathcal{D}_E$, there exists a predicate $p_1 : 2^E \rightarrow \{true, false\}$ and a predicate $p_2 : 2^{T_{\mathcal{D}_E}} \rightarrow \{true, false\}$ such that for and every candidate pattern P:

$$Select(P, \mathcal{D}_E) \equiv p_1(P) \wedge p_2(\mathcal{D}_E[\![P]\!]),$$

where $\mathcal{D}_E[\![P]\!]$ is the tid support set of P in $\mathcal{D}_E$.

Intuitively, the decomposability property discards selection criteria whose outcome depends on non local properties. For example, the following selection criterion: $Select(P, \mathcal{D}_E) \equiv true$ if $\exists e \in P : e \in \mathcal{D}_E(1)$ is not decomposable because the outcome depends on the content the transaction $\mathcal{D}_E(1)$ that may not belong to the reduced dataset of P.

To prove that ParaMiner is sound and complete for every decomposable selection criteria, we first prove the following lemmas.

Lemma 2 states that the closure Q of a pattern $P \cup \{e\}$ computed Line 4 in Algorithm 6 is correctly computed in $\mathcal{D}_P^{reduced}$ (i.e., $Clo(P \cup \{e\}, \mathcal{D}_E) = Clo(P \cup \{e\}, \mathcal{D}_P^{reduced})$).

Lemma 3 states that the *expand*() procedure of Algorithm 6 outputs the same set of patterns as *enum_clo*() given in Algorithm 2 which was proved to be sound and complete by (Boley et al, 2010).

These two lemmas are a corollary of the following technical lemma (Lemma 1) that results directly from Theorem 9.

Lemma 1 *If Select is decomposable then for every call to expand*$(P, \mathcal{D}_P^{reduced}, EL)$ *every element e not in EL, occurring in* $\mathcal{D}_P^{reduced}$*, and every* $S \subset \bigcap \mathcal{D}_E[P \cup \{e\}]$, $Select(P \cup \{e\} \cup S, \mathcal{D}_E) \equiv Select(P \cup \{e\} \cup S, \mathcal{D}_P^{reduced})$.

Lemma 2 *Let P be a closed pattern and* $\mathcal{D}_P^{reduced}$ *its reduced dataset. If e is an element such that* $P \cup \{e\}$ *is a pattern in* $\mathcal{D}_P^{reduced}$ *(e occurs in* $\mathcal{D}_P^{reduced}$ *and* $Select(P \cup \{e\}, \mathcal{D}_P^{reduced})$*) then, if the Select predicate is decomposable, the following property holds:* $Clo(P \cup \{e\}, \mathcal{D}_P^{reduced}) = Clo(P \cup \{e\}, \mathcal{D}_E)$.

Proof: We show that the generic closure operator provided in Algorithm 1 applied with $\mathcal{D}_P^{reduced}$ as the input dataset returns the same result than the one provided with $\mathcal{D}_E$.

We denote $Q(\mathcal{D}_E)$ the result of Algorithm 1 applied to $(P \cup \{e\}, \mathcal{D}_E)$ and $Q(\mathcal{D}_P^{reduced})$ the result of Algorithm 1 applied to $(P \cup \{e\}, \mathcal{D}_P^{reduced})$.

Let e' an element of $Q(\mathcal{D}_E)$, we show that e' is also in $Q(\mathcal{D}_P^{reduced})$.

e' is in $Q(\mathcal{D}_E)$ implies the following:

- $e' \in \bigcap \mathcal{D}_E[P \cup \{e\}]$
- let S be the set of elements added before e (i.e. in former iterations of the while loop in Line 4 in Algorithm 1. It is granted that $Select(P \cup \{e\} \cup S \cup \{e'\}, \mathcal{D}_E)$ is true.

Therefore if $e' \notin Q(\mathcal{D}_P^{reduced})$ either:

- $e' \notin \bigcap \mathcal{D}_P^{reduced}[P \cup \{e\}]$: this cannot be the case because e' occurs in $\mathcal{D}_P^{reduced}$ and $P \cup \{e\} \not\subseteq EL$ hence Theorem 9 applies with $Q = P$.
- there exists S' such that $Select(P \cup \{e\} \cup S' \cup \{e'\}, \mathcal{D}_P^{reduced})$ is false whereas $Select(P \cup \{e\} \cup S' \cup \{e'\}, \mathcal{D}_E)$ is true. This is impossible from Lemma 1 and $S = S' \cup \{e'\}$.

Conversely we show that if e' is an element of $Q(\mathcal{D}_P^{reduced})$, e' is also in $Q(\mathcal{D}_E)$.

If $e' \notin Q(\mathcal{D}_E)$ either:

- $e' \notin \bigcap \mathcal{D}_E[P \cup \{e\}]$: this cannot be the case because e' occurs in $\mathcal{D}_E$ and $P \cup \{e\} \not\subseteq EL$ hence Theorem 9 applies with $Q = P$.
- there exists S' such that $Select(P \cup \{e\} \cup S' \cup \{e'\}, \mathcal{D}_E)$ is false whereas $Select(P \cup \{e\} \cup S' \cup \{e'\}, \mathcal{D}_P^{reduced})$ is true. This is impossible from Lemma 1 and $S = S' \cup \{e'\}$.

Therefore $Q(\mathcal{D}_E) = Q(\mathcal{D}_P^{reduced})$. □

Lemma 3 *For a given dataset $\mathcal{D}_E$ and closure operator Clo, if the Select predicate is decomposable, then: for every argument $(P, \mathcal{D}_P^{reduced}, EL)$ of $expand(P, \mathcal{D}_P^{reduced}, EL)$, $expand(P, \mathcal{D}_P^{reduced}, EL) = enum\_clo(P, EL)$ when enum_clo is called with the set system $(E, \mathcal{F})$ formed by all the candidate patterns over E that occur in $\mathcal{D}_E$ and satisfy the selection criterion.*

Proof: Let Q be an output of $enum\_clo(P, EL)$: it is of the form $Clo(P \cup \{e\}, \mathcal{D}_E)$ where $P \cup \{e\}$ is a pattern and $e \notin EL$.

Suppose that it is not outputted by $expand(P, \mathcal{D}_P^{reduced}, EL)$. According to Algorithm 6 it means that:

- either e does not occur in $\mathcal{D}_P^{reduced}$ (Line 2 in Algorithm 6) which is impossible: since e does not belong to EL, it cannot have been suppressed by the dataset reduction returning $\mathcal{D}_P^{reduced}$.
- or $Select(P \cup \{e\}, \mathcal{D}_P^{reduced})$ is false (Line 3 in Algorithm 6). This cannot be the case if because of Lemma 1.
- or $Clo(P \cup \{e\}, \mathcal{D}_P^{reduced}) \cap EL \neq \emptyset$) (Line 5 in Algorithm 6). This cannot be the case since according to Lemma 2: $Clo(P \cup \{e\}, \mathcal{D}_P^{reduced}) = Clo(P \cup \{e\}, \mathcal{D}_E)$ and $Clo(P \cup \{e\}) \cap EL = \emptyset$ ($P \cup \{e\}$ is outputted by *enum_clo* only if $Clo(P \cup \{e\}) \cap EL = \emptyset$ according to Line 9 in Algorithm 2).

Conversely, we prove that a Q outputted by $expand(P, \mathcal{D}_P^{reduced}, EL)$ (Algorithm 6) is also outputted by $enum\_clo(P, EL)$ (Algorithm 2).

If Q is outputted in Line 7 in Algorithm 6 it means that there exists an e such that:

1. e occurs in $\mathcal{D}_P^{reduced}$ and $Select(P \cup \{e\}, \mathcal{D}_P^{reduced})$ is true, and
2. $EL \cap Clo(P \cup \{e\}, \mathcal{D}_P^{reduced}) = \emptyset$.

From 1., Lemma 1, the condition $Select(P \cup \{e\}, \mathcal{D}_E)$ Line 6 in Algorithm 2 is satisfied.

From 2., and Lemma 2 stating that $Clo(P \cup \{e\}, \mathcal{D}_P^{reduced}) = Clo(P \cup \{e\}, \mathcal{D}_E)$, the condition Line 9 in Algorithm 2 is also satisfied.

Therefore Q is outputted by $enu\_clo(P, EL)$ □

Theorem 10 (Soundness and completeness of ParaMiner**)** ParaMiner *computes the set of all closed patterns if the Select predicate is decomposable.*

Proof: It is a direct consequence of Lemma 3 and from the fact that *enum_clo*() (Algorithm 2) is a rephrasing of Boley et al.'s Algorithm 1 (Boley et al, 2010), which has been shown to compute the set of closed patterns.

We now prove that the *Select* predicate is decomposable for the FIM, CRG and GRI pattern mining problems introduced in Section 3. It guarantees that ParaMiner is sound and complete for these problems.

Corollary 1 ParaMiner *is sound and complete for the* FIM, CRG *and* GRI *problems.*

Proof: It is straightforward to show that the *Select* predicate for the FIM problem is decomposable, since $Select(P, \mathcal{D}_E)$ is true if and only if P is frequent in $\mathcal{D}_E$. Therefore $Select(P, \mathcal{D}_E) \equiv |\mathcal{D}_E[\![P]\!]| \geq \varepsilon$ (where $\mathcal{D}_E[\![P]\!]$ is the tid support set of P and ε is the frequency threshold).

Similarly, the *Select* predicate for CRG problem is decomposable because $Select(P, \mathcal{D}_E)$ is satisfied if and only if P is a connected graph and frequent in $\mathcal{D}_E$. Therefore $Select(P, \mathcal{D}_E) \equiv is\_connected(P) \wedge |\mathcal{D}_E[\![P]\!]| \geq \varepsilon$.

Finally let us consider the GRI problem. Let us recall that in this problem P is a pattern if and only if:

1. its first element is of the form $a^{\uparrow}$
2. and if its tid support set contains a tid *path* whose size is greater than ε.

Note that the first condition 1. can be expressed as a predicate over P and the condition 2. can be expressed as a predicate over the tid support set of P in $\mathcal{D}_E$. Hence *Select* is decomposable (Definition 16). □

5 Experimental study of parallel performances on large multi-core systems: impact of dataset reduction

In this section, we will present our experimental study on the parallel performances of ParaMiner. In particular we will study the impact of dataset reduction on parallel scalability. Previous research on efficient sequential pattern mining algorithms have shown that dataset reduction is a critical optimization to achieve efficient pattern testing (e.g., Han et al, 2000; Uno et al, 2004). However we will show in this section that systematically performing dataset reduction saturates the memory bus and leads to poor parallel scalability on large multi-core systems.

This observation lead us to study experimentally the *amortization* of each dataset reduction performed, and to define a criterion used to disable inefficient dataset reductions. We will show that it leads to an important increase in parallel scalability.

5.1 Experimental setting

We have conducted our preliminaries and final experiments on ParaMiner on assorted pattern mining problems (FIM, CRG and GRI) and datasets. For the FIM problem, we used two datasets from the FIMI repository commonly used to evaluate the performances of FIM algorithms (Goethals, 2004). We chose *Accidents* as a representative dense dataset and *BMS-WebView-2* as a representative sparse dataset, following experimentation protocol proposed by Uno et al (2005). For the CRG problem we used a real world gene network dataset called *Hughes-40*. This dataset is made of DAGs where each DAG represents a potential gene interaction network generated from *micro array experiments* (described by Imoto et al (2001)). This dataset has 1000 graphs

having in average 245 nodes and 270 edges. For the GRI problem, we used a real world dataset called $I4408$ which was also used by Do et al (2010) for their experiments. This dataset describes gene expression in the context of breast cancer. It has 100 records with 4408 attributes and is particularly challenging for existing algorithms. When it is necessary, we also use smaller datasets, namely $Mushroom$ for the FIM problem, $I500$ for the GRI problem, and *Hughes-60* for the CRG problem.

The specification of the datasets are given in Table 3. Note that these are the specifications of the *encoded* datasets as it was described in Section 3.

name	$\|\|\mathcal{D}_E\|\|$	$\|E\|$	Size (MiB)
BMS-2.	320,601	3,340	2.2
Accidents	11,500,870	468	34
Mushroom	186,852	119	0.55
I4408	50,985,072	4408	194.5
I500	5,824,224	500	22.2
Hughes-40	270,985	794	6.0
Hughes-60	173,131	665	4.1

Table 3 Characteristics of the datasets used in our experiments.

The specifications of multi-core system *Laptop* and *Server* that we chose to conduct our experiments are given in Table 4.

	Laptop	*Server*
# cores	4	32
Memory (GiB)	8	64
Processor type	Intel Core i7 X900	4 x Intel Core i7 X7560
Processor frequency (GHz)	2	2.27
Cache size (MiB)	8	4 × 24
Memory bus bandwidth (GiB/s)	7.9	9.7

Table 4 Specifications of the multi-core systems used: *Laptop* and *Server*.

Time measurements are conducted using the `time` Unix utility and include all the mining process from the creation of the unix process to its termination. This includes loading and pre-processing of the dataset, creation of the threads, mining the patterns, and storing them on the disk.

5.2 Parallel performance of ParaMiner: preliminary experiments

We begin our experimental study by evaluating the overall parallel performance of ParaMiner. To this end, we measure the *speedup* which is a well recognized metric to evaluate the ability of an algorithm to *scale* up with a large number of cores. The speedup is obtained by dividing the time required

for an execution on a single core, by the time required for an execution exploiting a number n of cores.

$$speedup_n = \frac{\text{execution time using 1 cores}}{\text{execution time using } n \text{ cores}}$$

A parallel algorithm has a good speedup with n cores if the speedup approaches n. We present in Figure 5 the speedups achieved by the implementation of ParaMiner presented in Section 4.1 for different pattern mining problems with a variable number of cores. The speedups achieved on *Laptop* are presented in Figure 5 (a) and the ones achieved on *Server* are presented in Figure 5 (b).

Speedups on Laptop (Figure 5 (a)): When exploiting the 4 cores available on *Laptop*, ParaMiner performs 4 times faster for FIM on *BMS-WebView-2*, as well as for CRG and GRI. On *Accidents*, the speedup is 3 due to long loading time of the *Accidents* which is performed sequentially in ParaMiner. However this is constant time at the beginning of the execution and does not further impact the efficiency of the mining process.

The last column in Figure 5 (a) labeled `w/ht` shows the speedup achieved with the *hyper-threading* technology activated. *Hyper-threading* enables *instruction level parallelism* allowing a single core to perform multiple instructions simultaneously. ParaMiner is able to further benefit from this technology and achieve a significant improvement resulting in a super-linear speedup of 5 on 4 cores.

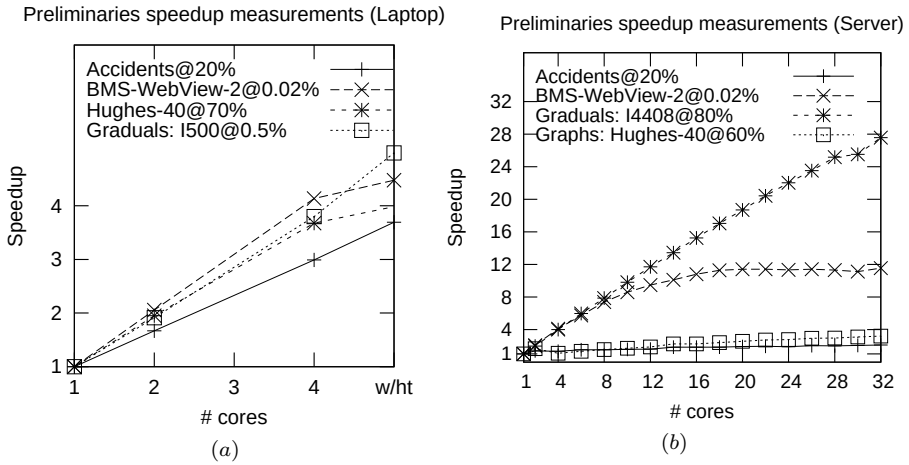

Fig. 5 Preliminaries experiments: Speedups on *Laptop* (a), and on *Server* (b).

Speedups on Server (Figure 5 (b)): On *Server*, the maximum speedup varies depending on problems and datasets at hand. The speedup is near-linear for gradual itemset mining on $I4408$ (28 on 32 cores), but is below 4 on 32 cores for mining frequent itemsets on the *Accidents* dataset. This indicates an inadequate exploitation of the architecture.

We conduct further experimentation to identify the root of the performance decrease. Modern processors embed a *performance monitor unit* (PMU) that permits us to conduct lower level measurements and detect abnormal behaviors, in particular memory bus overflow. When the bus is overflowed, memory operations are delayed, and cores have to wait which ultimately leads to bad speedups. In Figure 6 we have compared the proportion of delayed memory operations along two execution of ParaMiner: one on the gri problem that exhibits good speedups, and one on the fim problem that exhibits lower speedups.

We observe that for the gri problem, the proportion of delayed memory operations remains constant whatever the number of cores in exploitation. For the fim problem, the proportion of delayed memory operations increases significantly when ParaMiner is executed with more cores. On *Server*, typical accesses to the memory are performed in more than 128 cycles, (less if the data accessed is in cache memory). When running ParaMiner on fim with 32 cores, the proportion of memory operations performed in more than 128 cycles goes from 0% to 3% and a significant number of memory operations are delayed over 512 cycles. Figure 6 shows that there is a severe memory bus overflow for the fim problem. This bus overflow is the main cause of the non optimal speedups observed in Figure 5.

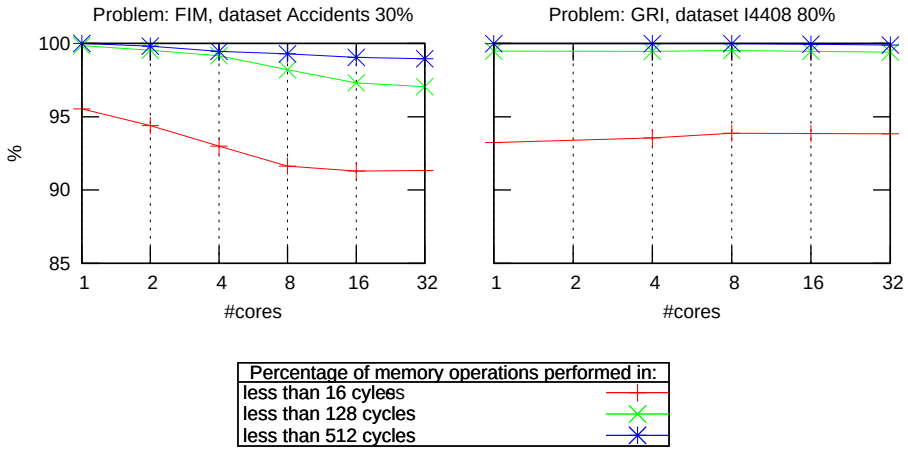

Fig. 6 Memory transfers latencies

5.3 Impact of dataset reduction on parallel efficiency

Memory bus overflow is typically caused by a large number of memory accesses in a short period of time, exceeding bus bandwidth. In pattern mining algorithms, large scale memory operations such as dataset reductions perform an important amount of memory accesses. However efficient dataset reductions also drastically reduce the size of the datasets thus contributing to significantly decrease the amount of memory operations required for the rest of the mining process. This is an amortization problem: the reductions that cost a lot of memory accesses without bringing sufficient reduction of the dataset must be avoided.

In order to measure efficiency of the reductions, we introduce the *reduction factor*. The reduction factor compares the memory space required to store a reduced dataset to the memory space required to store the relevant transactions in the parent dataset. Note that identical transactions are stored only once, therefore the size occupied by a dataset in memory is bounded by the number of distinct transactions.

Definition 17 (Reduction factor of a dataset) Let P and Q be two closed pattern such that Q is a descendant of P in the enumeration tree. Let also $\mathcal{D}_P^{reduced}$ be the reduced dataset of P and $\mathcal{D}_Q^{reduced}$ be the reduced dataset of Q. The reduction factor achieved by $\mathcal{D}_Q^{reduced}$ is equal to the ratio between the number of distinct transactions in $\mathcal{D}_P^{reduced}[Q]$ and the number of distinct transactions in $\mathcal{D}_Q^{reduced}$.

We have instrumented an implementation of ParaMiner in order to measure the reduction factor achieved by every reductions during an execution of ParaMiner on *Accidents*. Thanks to the reduction factors obtained we could conduct a new series of experiments: for each experiment a percentage x of the total number of reduction is set. Only the x% of the reductions that have the highest reduction factors will be performed. During these experiments we measure the number of memory accesses performed per thousands of instructions. This measure corresponds to the requested bus bandwidth.

The results of this experiments are presented in Figure 7.
Figure 7 shows that:

1. Between performing no reductions and performing 12.5% of the reductions with the higher reduction factor, the number of memory accesses per thousands instructions drops by more than three times. These reductions are amortized since they contribute to the overall diminution of the memory accesses.
2. Conversely, the 12.5% reductions with the lowest reduction factors (rightmost bar on the figure) generate an important increase in the number of memory accesses per thousands of instructions. They introduce more overhead than they contribute to reduce the number of dataset accesses.

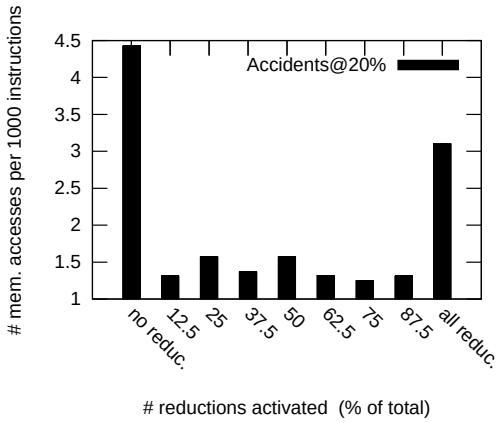

Fig. 7 Efficiency of dataset reduction

5.4 Amortizing the cost of dataset reduction

In order to improve ParaMiner's parallel scalability, the previous results suggest that reductions not amortized should be deactivated. We have also seen that these reductions have a low reduction factor. However, during an execution of ParaMiner, it is not possible to know in advance the reduction factor without an important computational overhead.

In the following, we conduct a brief theoretical study and provide an upper bound to the reduction factor that can be achieved. This upper bound can be used to deactivate dataset reductions whose reduction factors are low.

Since the size required to store a dataset depends on the number of distinct transactions, we provide an upper bound for the number of distinct transactions in a dataset.

Proposition 1 (Maximum number of distinct transactions) *Given a ground set E and an exclusion list EL, let $\mathcal{D}_P^{reduced}$ be the reduced dataset of a closed pattern P. The number of distinct transactions in $\mathcal{D}_P^{reduced}$ is lower or equal than $2^{|E \setminus EL|}$.*

Proof: According to the Algorithm 7, after a reduction each transaction contains the same set of items, thus the number of distinct transactions after a reduction is equal to the number of partitions. We recall that two transactions t_1 and t_2 fall in the same partition if they contain the same set elements not included in EL: i.e. if $t_1 \setminus EL = t_2 \setminus EL$. Therefore there cannot be more distinct transactions than there exist subsets of the set $E \setminus EL$. The number of subsets of $E \setminus EL$ is equal to $2^{|E \setminus EL|}$, thus the maximum number of distinct transactions is also $2^{|E \setminus EL|}$. □

In ParaMiner, every reduction operates on a parent dataset and results in a child dataset. Let $\mathcal{D}_P^{reduced}$ and $\mathcal{D}_Q^{reduced}$, be a parent and a child dataset

respectively. Note that both datasets are reduced datasets, resulting of a call to *reduce*(). We call EL_P the exclusion list parameter of the *reduce*() call resulting in $\mathcal{D}_P^{reduced}$ and EL_Q the exclusion list parameter of the *reduce*() call resulting in $\mathcal{D}_P^{reduced}$. We now introduce the notion of *exclusion list tail* abbreviated EL_{tail} which represents the set of elements added to the exclusion list since the last reduction performed.

Definition 18 (Exclusion list tail) Given a parent reduced dataset $\mathcal{D}_P^{reduced}$ reduced with an exclusion list EL_P and a child reduced dataset reduced with an exclusion list EL_Q, we call *exclusion list tail* the set of elements added after the reduction of $\mathcal{D}_P^{reduced}$: $EL_{tail} = EL_Q \setminus EL_P$.

The exclusion list tail can be used to compute the maximum reduction factor that can be achieved between two reductions.

Proposition 2 (Maximum reduction factor) *The maximum reduction factor that can be achieved is* $2^{|EL_{tail}|}$

Proof: For the sake of clarity, we denote by $\#trans(dataset)$ the number of distinct transactions in *dataset*.

The minimum number of distinct transactions in $\mathcal{D}_Q^{reduced}$ is the minimum number of partitions produced by the *partition*() function applied to $\mathcal{D}_P^{reduced}$. Two transactions t_1 and t_2 from $\mathcal{D}_P^{reduced}$ belong to the same partition in $\mathcal{D}_Q^{reduced}$ if and only if $t_1 \setminus EL_Q = t_2 \setminus EL_Q$ that is $t_1 \setminus (EL_P \cup EL_{tail}) = t_2 \setminus (EL_P \cup EL_{tail})$. However, since $\mathcal{D}_P^{reduced}$ is also a reduced dataset, for every two transactions t_1, t_2: $t_1 \setminus EL_P \neq t_2 \setminus EL_P$. Hence two transactions t_1 and t_2 in $\mathcal{D}_P^{reduced}$ belong to the same partition in $\mathcal{D}_Q^{reduced}$ if and only if $t_1 \setminus EL_{tail} = t_2 \setminus EL_{tail}$. Therefore every two transactions in a partition can only differ by a subset of elements of EL_{tail}. Since the number of distinct subsets of EL_{tail} is at most $2^{|EL_{tail}|}$, the maximum number of distinct transactions in a partition is at most $2^{|EL_{tail}|}$.

It follows that the minimum number of partitions is at least $\frac{\#trans(\mathcal{D}_P^{reduced})}{2^{|EL_{tail}|}}$ and that the minimum number of distinct transactions in $\mathcal{D}_Q^{reduced}$ is also at least $\frac{\#trans(\mathcal{D}_P^{reduced})}{2^{|EL_{tail}|}}$. Therefore $\frac{\#trans(\mathcal{D}_P^{reduced})}{\#trans(\mathcal{D}_Q^{reduced})} \leq 2^{|EL_{tail}|}$. Since $\mathcal{D}_P^{reduced}[Q]$ is a subset of $\mathcal{D}_P^{reduced}$, the reduction factor $\frac{\#trans(\mathcal{D}_P^{reduced}[Q])}{\#trans(\mathcal{D}_Q^{reduced})}$ is at most $2^{|EL_{tail}|}$. □

To take into account this result, we have modified the reduction algorithm (Algorithm 7). In the new algorithm Algorithm 9 the *EL-reduction* is not performed when the maximum reduction factor is below a user defined threshold $\delta > 1$. It is worth noting that when the *EL-reduction* is not performed, each transaction of the child dataset occurs in the parent dataset, therefore no new additional memory space is required and the reduction is costless. With this modification we avoid costly dataset reductions with the lowest reduction factors.

Algorithm 9 The modified dataset reduction algorithm

1: $reduce(\mathcal{D}_P^{reduced}, e, EL)$
Require: The reduced dataset $\mathcal{D}_P^{reduced}$ of the parent P of Q, the augmenting element e such that $Q = Clo(P \cup \{e\}, \mathcal{D}_E)$, the exclusion list EL, a minimum reduction factor threshold δ.
Ensure: Returns the reduced dataset of Q: $\mathcal{D}_Q^{reduced}$
2: $\mathcal{D}_Q^{reduced} \leftarrow \mathcal{D}_P^{reduced}[\{e\}]$
3: **if** $2^{EL\_tail} \geq \delta$ **then**
4: **for all** $G \in partition(\mathcal{D}_Q^{reduced}, EL)$ **do**
5: **for all** $e \in EL$ **do**
6: **if** there exists $t' \in G$ such that $e \notin t'$ **then**
7: Suppress e from all the transactions in G
8: **end if**
9: **end for**
10: **end for**
11: **end if**
12:
13: **return** $\mathcal{D}_Q^{reduced}$

In Figure 8, we present the speedup of the modified ParaMiner on *Server*. For the GRI problem the speedup remain near optimal. In the case of the CRG problem, up to twelve cores there is a large increase of speedup. However it does not scale well using more than twelve cores. The reason is that reduction factors achieved on this dataset are not very good: for less than twelve cores, the bus provides enough bandwidth with the reduction at hand but is saturated again when exploiting more cores. For the FIM problem, the speedups are significantly better than in Figure 5 especially for *Accidents* which speedup has been multiplied by more than 5. This results show that our criterion for deactivating dataset reduction allows to improve the parallel scalability.

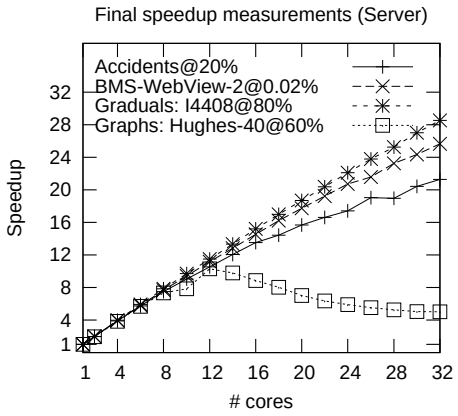

Fig. 8 Final speedups on *Server*.

6 Comparative experiments with specialized mining algorithms

In this section, we compare ParaMiner's efficiency to state of the art specialized algorithms. We first compare the execution times necessary to mine closed frequent itemsets. In this experiment ParaMiner is compared with the state-of-the-art parallel ad-hoc algorithms PLCM (Negrevergne et al, 2010) and MT-Closed (Lucchese et al, 2007). These algorithm are parallel implementations of the two fastest algorithm according to the FIMI workshop(Goethals, 2004). The execution times and the memory usage are reported for the sparse dataset *BMS-WebView* and the dense dataset *Accidents* in Figure 9 (*a*) and (*b*).

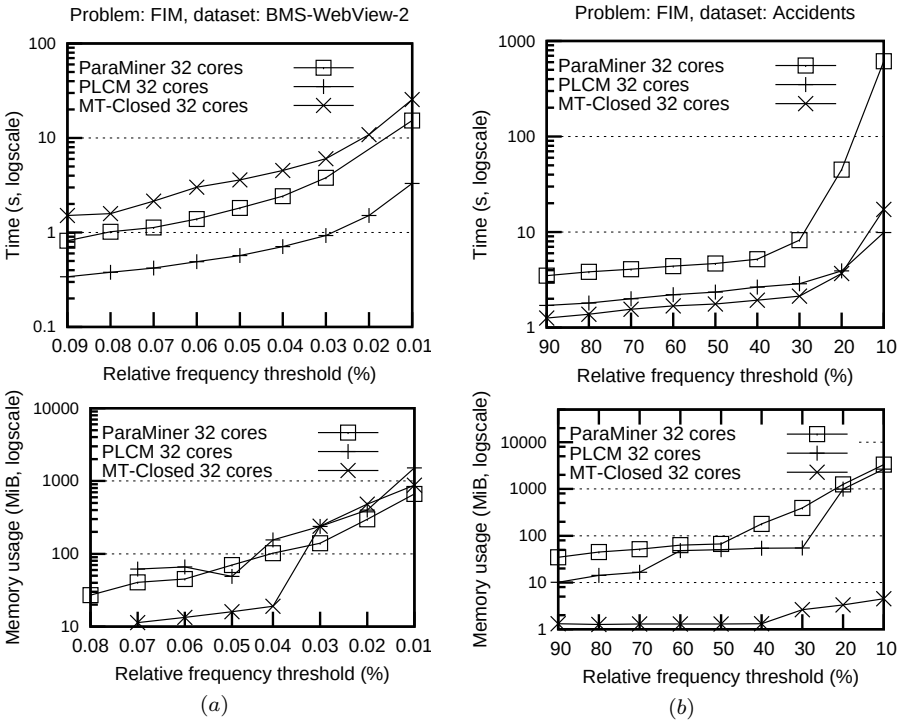

Fig. 9 Comparative experiments for FIM, on *BMS-WebView-2*: (*a*) and *Accidents*: (*b*). Runtimes (top) and memory usage (bottom).

On the sparse dataset, ParaMiner have shorter execution times than MT-Closed and 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. It is worth noting that memory usage for MT-Closed is below other algorithms because it uses bitmap representations that are more efficient on dense datasets.

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 execution times.

CRG *problem:* Even though relational graph datasets are common in bio informatics and social networks, there is no implementation available that is dedicated for mining subgraphs in relational graphs datasets. The only way to mine subgraphs in this type of datasets is thus to use more general graph mining algorithms such as GSPAN (Yan and Han, 2002) or GASTON (Nijssen and Kok, 2004). Comparing PARAMINER with these algorithm is not fair, however GSPAN and GASTON are the most efficient algorithms that a practitioner can find to analyze a relational graph dataset. Mining times and memory usage for the three algorithms running on *Hughes-60* dataset are presented in Figure 10 (*a*). Since GSPAN and GASTON are have not been paralleled, we run PARAMINER on a single core.

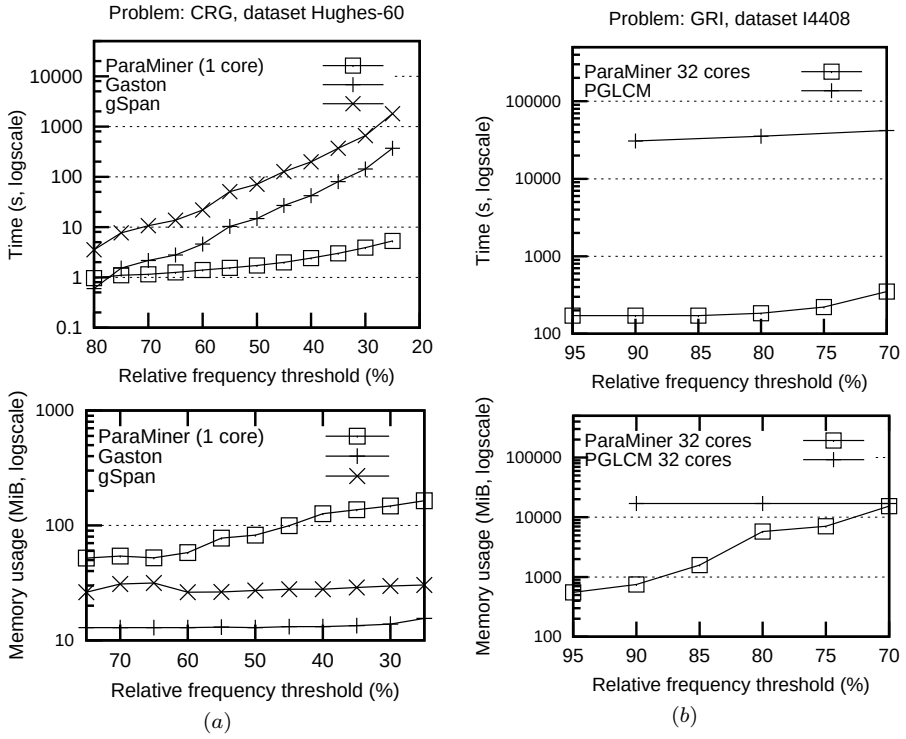

Fig. 10 Comparative experiments for CRG on Hughes-40 and GRI on I4408. Runtimes (top) and memory usage (bottom).

It appears clearly that neither GSPAN nor GASTON can scale efficiently on this dataset. PARAMINER which was parametrized with the accurate problem

definition can complete the mining task within much shorter times. This experiment clearly demonstrates that the adaptability of ParaMiner can be very beneficial to efficiency and shows the limit of non-parametrizable algorithms.

gri *problem:* We compared ParaMiner with state-of-the-art parallel algorithm Pglcm (Do et al, 2010). We have use *I4408*, a real gene expression dataset with 100 records and 4408 attributes. *I4408* was the largest dataset used to evaluate the performances of Pglcm. The results are shown in Figure 10(*b*).

Comparing ParaMiner with Pglcm shows that ParaMiner is two orders of magnitude faster than Pglcm. This is mostly due to the lack of dataset reduction in Pglcm. As a consequence it needs more than ten hours of computation whereas ParaMiner completes in few minutes. It is worth noting that designing a dataset reduction for gradual itemsets was left as an open problem by Do et al (2010), this problem is now solved.

7 Related work

Our framework is able to capture pattern mining problems by encoding the set of meaningful patterns as sets satisfying a selection criterion in a set system. Constraint based itemset mining is another framework which focuses on extracting itemsets satisfying constraints expressed over patterns in the manner of our selection criterion. Altough the two approaches are different, the resulting frameworks and algorithms are quite comparable. Indeed some problems can be formulated in both frameworks. We show that there exist strong connections between our approach and the constraint based approach, then we go further and study the accessibility of some constraint-based itemset mining problems.

Other generic approaches for pattern mining can be classified in the category of *toolbox approaches*. These approaches are based on the observation that most pattern mining algorithms share similar structures and try to define a set of algorithmic tools that could be use to create easily new pattern mining algorithms. Altough they cannot be directly compared with ParaMiner, we discuss their expressivity and efficiency in Section 7.1.

Because we have a strong focus on parallel pattern mining, we also present various works conducted on this topic. Most of those works clearly demonstrate the difficulty of designing parallel pattern mining algorithm. We briefly discuss the issues identified and the solutions proposed by these publications in Section 7.3.

7.1 Constraint based itemset mining

The major approach to tackle generic pattern mining is constraint-based pattern mining which has been first proposed by Srikant et al (1997). With this

approach meaningful patterns are defined through constraints expressed over patterns.

Constraint pattern mining aims at classifying existing constraints into constraint classes such as *anti-monotone* constraints (Mannila and Toivonen, 1997), *monotone* constraints (Mannila and Toivonen, 1997), *succinct* constraints (Ng et al, 1998), *convertible* constraints (Pei and Han, 2000; Pei et al, 2001), *loose anti-monotone* constraints (Bonchi and Lucchese, 2007) and the recent *area* constraint (Soulet and Crémilleux, 2005).

In order to build efficient constraint based mining algorithms, a large body of work has been dedicated to push the constraints as deeply as possible in the algorithms in order to prune large portions of the search space and dramatically reduce the mining times. This approach is comparable to our work in the sense that it studies properties of the search space in order to design algorithms whose soundness does not depend on specific properties of the pattern mining problem at hand but on more general properties.

Connections between constraints and accessibility properties:
It can be demonstrated that a set of patterns defined with an anti-monotone constraint has a corresponding set system that is independent. However, aside from anti-monotony, we have shown that none of the set systems associated to the existing classes of constraint verifies the accessibility property (Negrevergne, 2011). Indeed, these constraints do not guarantee that the empty set is part of the set of patterns, which is a requirement for accessibility.

The fact that the set of patterns does not contain the empty set does not mean that it does not exhibit structural properties similar to accessibility *locally*. We thus propose to amend to notion of accessibility to *partial accessibility*. In order to define partial accessibility, we use the concept of *lower border* which will represent the smallest patterns satisfying the constraints at hand. It was introduced by Sun and Yu (2007) for itemsets. We generalize below their definition for set systems.

Definition 19 (Lower border of a set system) Let $(E, \mathcal{F})$ be a set system. The lower border of $\mathcal{F}$, denoted $BD^-_{\mathcal{F}}$, is the set of patterns such that:

i. $BD^-_{\mathcal{F}} \subseteq \mathcal{F}$
ii. For any two patterns $X, Y \in BD^-_{\mathcal{F}}$, $X \not\subseteq Y$ and $Y \not\subseteq X$ ($BD^-_{\mathcal{F}}$ is called an antichain)
iii. For any pattern $X \in \mathcal{F}$, there exists at least one pattern $Y \in BD^-_{\mathcal{F}}$ such that $Y \subseteq X$.

We can now propose a weaker property of accessibility based on the lower border.

Definition 20 (Partial accessibility) Let $(E, \mathcal{F})$ be a set system. $(E, \mathcal{F})$ is partially accessible if for every $X \in \mathcal{F} \setminus BD^-_{\mathcal{F}}$ there exists some $e \in X$ such that $X \setminus \{e\} \in \mathcal{F}$.

Definition 21 (Partial strong accessibility) Let $(E, \mathcal{F})$ be a set system. $(E, \mathcal{F})$ is partially strongly accessible if it is partially accessible and if for every $X \subset Y$ with $X \in \mathcal{F}$ and $Y \in \mathcal{F} \setminus BD^-_{\mathcal{F}}$, there exists some $e \in Y \setminus X$ such that $X \cup \{e\} \in \mathcal{F}$.

We have already shown that the set systems corresponding to a set of patterns defined with a *monotone* constraint, with a *convertible anti-monotone* constraint or with a *loose anti-monotone* constraint (Negrevergne, 2011) is partially accessible. Studying the connection between accessibility properties and constraint based pattern mining opens some important research perspectives that will be discussed in Section 8. Note that, in the context of graph mining Zhu et al (2007) have proposed properties on constraints called *strong p-anti-monotonicity* and *weak p-anti-monotonicity* that correspond to independence and accessibility of set systems. This work is interesting as it is a first step toward extending ParaMiner to patterns more complex than sets.

ParaMiner *vs other constraint-based pattern mining algorithms:*
Even though the two approaches are comparable, there are some important distinctions between ParaMiner's and constraint-based pattern mining algorithms. First, many constraint based algorithms such the one proposed by Bonchi and Lucchese (2007) focus on extracting *frequent* patterns with additional constraints. This is a major difference with ParaMiner because it prevents from using the algorithm to solve pattern mining problems using a non trivial encoding such as the one we used to mine gradual itemsets.

Another important distinction between the approaches is that constraint based approaches tend to see the closure as another constraint. As pointed by Bonchi and Lucchese (2004), combining closure with other constraints can lead to ambiguous problem definitions. As a consequence, there is little work on closed constraint-based pattern mining. However it is worth mentioning the work by Guns et al (May 2011) with their CP4IM which is able to efficiently combine the closure constraint with other constraints, and mine closed pattern such as closed frequent itemsets and closed discriminative itemsets fairly efficiently.

As a consequence, their algorithm outperform other constraint based algorithms. In the experiments presented by Guns et al (May 2011), CP4IM set up for closed frequent itemset mining exhibit execution times within two orders of magnitude with the state of the art closed specialized algorithm LCM on the small dataset $Mushroom$.

For the sake of completeness, we compare the execution times of with ParaMiner's on this dataset. The execution times are obtained by running both algorithms on *Server* using only one core (thus not exploiting parallelism). The results are reported in Figure 11.

These results show that on this $Mushroom$, CP4IM is more than one order of magnitude slower than ParaMiner. On larger datasets such as $BMS-WebView-2$ or *Accident* datasets CP4IM cannot not complete due to memory exhaustion despite the 64 GiB available on *Server*.

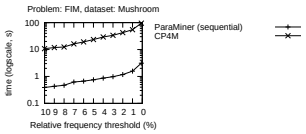

Fig. 11 Comparative experiments ParaMiner vs CP4IM. Sequential execution on *Mushroom*.

7.2 Other generic approaches

Other approaches for generic pattern mining have been proposed and are different from constraint-based pattern mining. One of them is DMTL which stands for *Data Mining Template Library* (Chaoji et al, 2008). It is restricted to frequent patterns, but it can handle patterns having structures of sequences, trees or graphs using complex isomorphism tests for testing pattern inclusion in the dataset. This is something that is not possible neither with ParaMiner nor with constraint-based pattern mining approaches. However, DMTL cannot mine closed patterns, and is based on legacy algorithm that do not handle efficient dataset reductions. It is thus orders of magnitude slower than current state of the art algorithms and do not scale well on real-world datasets. Another approach is the IZI library (Flouvat et al, 2009). It deals with patterns representable as sets. The patterns are specified through a predicate, similar to our selection criterion and to what is done in constraint-based pattern mining. This predicate must be monotone or anti-monotone. One of the interests of IZI compared to ParaMiner is that it can give the upper- and lower-border of the results, which can be of interest in some applications. However IZI does not handle closure and is based on the Apriori algorithm. It is thus very slow and can only be used on small synthetic datasets.

7.3 Parallel pattern mining

Exploiting parallel computers to tackle larger datasets is not a new idea. Parallel algorithms have been quickly considered as a solution to reduce execu-

tion times due to the heavy computational requirements of pattern mining (e.g. Agrawal and Shafer, 1996). With the generalization of multi-core processors, having efficient parallel pattern mining algorithms became a necessity. However parallel pattern mining has been shown to be a challenging problem because of several issues well known by the parallel computing community, but particularly intense when dealing with pattern mining algorithms.

Buehrer et al (2006) proposed a parallelization of the gSpan graph mining algorithm. They observe that naive task decompositions generally create tasks of very irregular size which provoke *load imbalance* among the cores and eventually impact the performances. They proposed a task decomposition which is able to dynamically create new tasks when the work is not correctly balanced among the cores. The task decomposition in ParaMiner is recursive, which allows larger tasks to be decomposed in sub-tasks. In previous work (Negrevergne, 2011) we conducted experiments that demonstrated that load imbalance was not an issue in ParaMiner.

For some pattern mining problems, ensuring correct load balancing is insufficient to reach the theoretical performances of the computing platform. This is visible in Figure 5. Ghoting et al (2005) have shown that pattern mining algorithms generally perform more accesses to the memory than others which put too much pressure on the bus connecting memory and computing cores (Tatikonda and Parthasarathy, 2009). In addition, traditional optimizations tend to worsen to problem (Negrevergne et al, 2010) because they only reduce the amount of computation required and not the amount of data transfers. This problem can be solved by performing deep changes in the algorithms to reduce data transfers generally at the cost of an additional computations. Although these publications present studies conducted on other pattern mining algorithms, they all illustrate common issues of parallel pattern mining. They have inspired our study of dataset reduction in parallel and also propose interesting ideas to further improve ParaMiner's parallel performances (See following Section 8).

8 Conclusion

In this paper, we have proposed an efficient generic and parallel pattern mining algorithm called ParaMiner. ParaMiner represents patterns and datasets using sets and focuses on problems whose corresponding set system is strongly accessible and selection criterion is decomposable. We have shown that this covers many interesting pattern mining problems while allowing short mining times. This makes ParaMiner a platform that can be parametrized to mine new types of patterns efficiently.

Efficient pattern mining relies on the fundamental principle that one can reduce *the pattern search space* and the *data search space* simultaneously. Our main contribution permits to reduce the data search space thanks to a novel dataset reduction named *EL-reduction*, that can be applied to a broad range of pattern mining problems, unlike previous work. We also presented the first

study of the impact of dataset reduction on parallel execution, showing that all dataset reductions were not equally useful: some are unavoidable whereas others congest the memory bus without reducing computation times. This study has allowed us to improve ParaMiner's parallel scalability.

This makes ParaMiner adequate for an intermediary class of pattern mining problems that is in between the class of problems that are variations of frequent itemset mining, and the class of structured pattern mining problems (sequences, trees and graphs) that can be encoded in variations of sequence mining problems. This class captures the problems in which patterns can be encoded by sets with simple constraints (i.e. constraints guaranteeing that the selection predicate is monotonic with respect to set inclusion restricted to the sets that represent patterns). This intermediate class has received less attention, despite the fact that many practical pattern mining problems, such as the CRG problem and the GRI problem studied in this paper, fall in it.

Finally, we recall that ParaMiner is available as an open source software on the authors' web page[1].

Future work

There exists a variety of work that can be conducted to improve ParaMiner. We distinguish two types of works: the ones to improve ParaMiner's efficiency, and the ones to make it work for a broader range of pattern mining problems.

Many research works on parallel pattern mining algorithms have shown that pattern mining is a challenging problem for parallelism due to its very intensive memory usage. There is still room for improvement in this direction in order to scale up to upcoming platforms. One can implement *tiling* techniques such as the ones proposed by Ghoting et al (2005). These techniques improve the memory locality and reduce the number of costly accesses to the memory. In ParaMiner it would consist in breaking up the larger datasets into chunks, and performing all the computations on a chunk before moving to the next one.

Although we have focused on multi-core architectures, it is worth noting that ParaMiner can be modified to exploit larger parallel platforms such as clusters of computers. Thanks to our decomposition of the search space into independent tasks and the dataset reduction technique no task migration is required *a priori*. In practice however, it can be required to balance the work on clusters with thousands of nodes.

In this paper, we have demonstrated the genericity of ParaMiner on three pattern mining problems, it has been shown by Arimura and Uno (2009) that many other pattern mining problems such as rigid itemset sequences or picture patterns can be represented as sets in a set system. For most of these problems, the accessibility of the corresponding set system have been proved. However it has not yet been demonstrated whether or not they are *strongly*

[1] http://membres-liglab.imag.fr/negrevergne/

accessible and if they satisfy the decomposability property. For most of these problems there exists no available algorithm. Hence proving that they admit an encoding adequate to ParaMiner would be a major benefit for the pattern mining community.

In order to improve ParaMiner's genericity we have proposed the notion of partial (strong) accessibility. We have shown that this notion permits to express the accessibility property of many constraint based pattern mining problems. We are currently working on ParaMiner to make it able to mine sets of patterns which are only partially strongly accessible.

Other problems such as the general graph mining problem cannot be described in the set framework efficiently. However Zhu et al (2007) have proposed properties similar to accessibility for sets of graphs. In this paper we demonstrated that *strong* accessibility was crucial to achieve efficiency. A natural extension of Zhu et al (2007)'s work would be to define strong accessibility for graphs and exploit its properties to build efficient generic algorithms for general graph mining.

References

Agrawal R, Shafer J (1996) Parallel mining of association rules. Knowledge and Data Engineering, IEEE Transactions on 8(6):962–969

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

Arimura H, Uno T (2009) Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In: SDM, pp 1087–1098

Ayouni S, Laurent A, Yahia SB, Poncelet P (2010) Mining closed gradual patterns. In: ICAISC, pp 267–274

Boley M, Horváth T, Poigné A, Wrobel S (2010) Listing closed sets of strongly accessible set systems with applications to data mining. Theor Comput Sci 411(3):691–700

Bonchi F, Lucchese C (2004) On closed constrained frequent pattern mining. In: ICDM, pp 35–42

Bonchi F, Lucchese C (2007) Extending the state-of-the-art of constraint-based pattern discovery. Data Knowl Eng 60(2):377–399

Buehrer G, Parthasarathy S, Chen YK (2006) Adaptive parallel graph mining for cmp architectures. In: ICDM, pp 97–106

Chaoji V, Hasan MA, Salem S, Zaki MJ (2008) An integrated, generic approach to pattern mining: data mining template library. Data Min Knowl Discov 17(3):457–495

Di-Jorio L, Laurent A, Teisseire M (2009) Mining frequent gradual itemsets from large databases. Advances in Intelligent Data Analysis VIII pp 297–308

Do TDT, Laurent A, Termier A (2010) Pglcm: Efficient parallel mining of closed frequent gradual itemsets. In: ICDM, pp 138–147

Flouvat F, Marchi FD, Petit JM (2009) The izi project: Easy prototyping of interesting pattern mining algorithms. In: PAKDD Workshops, pp 1–15

Ghoting A, Buehrer G, Parthasarathy S, Kim D, Nguyen A, Chen YK, Dubey P (2005) Cache-conscious frequent pattern mining on a modern processor. In: Very Large Data Bases (VLDB), VLDB Endowment, pp 577–588

Goethals B (2004) Fimi repository website. `http://fimi.cs.helsinki.fi/`

Guns T, Nijssen S, Raedt LD (May 2011) Itemset mining: A constraint programming perspective. Artificial Intelligence

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. Special Interest Group on Managment of Data (SIGMOD) 29(2):1–12

Imoto S, Goto T, Miyano S (2001) Estimation of genetic networks and functional structures between genes by using Bayesian networks and nonparametric regression. In: PSB'02: Kauai, Hawaii, 3-7 January 2002, World Scientific Pub Co Inc, p 175

Lucchese C, Orlando S, Perego R (2007) Parallel mining of frequent closed patterns: Harnessing modern computer architectures. In: ICDM, pp 242–251

Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258

Negrevergne B (2011) A generic and parallel pattern mining algorithm for multi-core architectures. PhD thesis, University of Grenoble

Negrevergne B, Termier A, Mehaut JF, Uno T (2010) Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses. In: HPCS, pp 521–528

Ng RT, Lakshmanan LVS, Han J, Pang A (1998) Exploratory mining and pruning optimizations of constrained association rules. In: SIGMOD Conference, pp 13–24

Nijssen S, Kok J (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 647–652

Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: ICDT, pp 398–416

Pei J, Han J (2000) Can we push more constraints into frequent pattern mining? In: KDD, pp 350–354

Pei J, Han J, Lakshmanan LVS (2001) Mining frequent item sets with convertible constraints. In: ICDE, pp 433–442

Soulet A, Crémilleux B (2005) An efficient framework for mining flexible constraints. In: PAKDD, pp 661–671

Srikant R, Vu Q, Agrawal R (1997) Mining association rules with item constraints. In: KDD, pp 67–73

Sun X, Yu PS (2007) Hiding sensitive frequent itemsets by a border-based approach. JCSE 1(1):74–94

Tatikonda S, Parthasarathy S (2009) Mining tree-structured data on multicore systems. In: VLDB, pp 694–705

Uno T, Asai T, Uchida Y, Arimura H (2003) Lcm: An efficient algorithm for enumerating frequent closed item sets. In: Proc. IEEE ICDM, Citeseer, vol 3

Uno T, Kiyomi M, Arimura H (2004) Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In: FIMI

Uno T, Kiyomi M, Arimura H (2005) Lcm ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In: OSDM'05 Workshop, ACM, pp 77–86

Yan X, Han J (2002) gspan: Graph-based substructure pattern mining. International Conference on Data Mining (ICDM)

Yan X, Zhou XJ, Han J (2005) Mining closed relational graphs with connectivity constraints. In: ICDE, pp 357–358

Zhu F, Yan X, Han J, Yu P (2007) gprune: a constraint pushing framework for graph pattern mining. Advances in Knowledge Discovery and Data Mining pp 388–400