PADS: A Simple Yet Effective Pattern-Aware Dynamic Search Method for Fast Maximal Frequent Pattern Mining

Xinghuo Zeng[†] Jian Pei[†] Ke Wang[†] Jinyan Li[‡]

[†] School of Computing Science, Simon Fraser University, Canada. {xzeng, jpei, wangk}@cs.sfu.ca

[‡] School of Computer Engineering, Nanyang Technological University, Singapore. jyli@ntu.edu.sg

Abstract

While frequent pattern mining is fundamental for many data mining tasks, mining maximal frequent patterns efficiently is important in both theory and applications of frequent pattern mining. The fundamental challenge is how to search a large space of item combinations. Most of the existing methods search an enumeration tree of item combinations in a depth-first manner.

In this paper, we develop a new technique for more efficient max-pattern mining. Our method is pattern-aware: it uses the patterns already found to schedule its future search so that many search subspaces can be pruned. We present efficient techniques to implement the new approach. As indicated by a systematic empirical study using the benchmark data sets, our new approach outperforms the currently fastest max-pattern mining algorithms FPMax* and LCM2 clearly.

1 Introduction

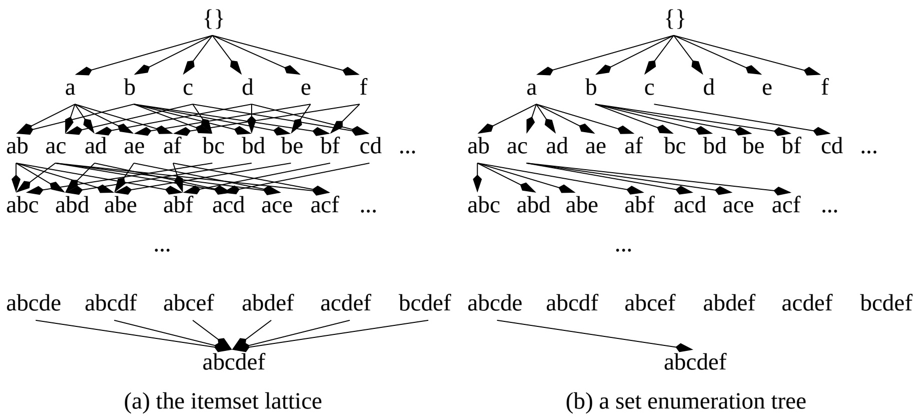
Let I be a set of items. An *itemset* X is a subset of I. A *transaction* is a tuple (tid, Y) where tid is a unique transaction-id and Y is an itemset. Transaction (tid, Y) is said to *contain* itemset X if $X \subseteq Y$. For a given *transaction database* TDB which consists of a set of transactions, the *support* of an itemset X is the number of transactions containing X, that is, $sup(X) = |\{(tid, Y) \in TDB | X \subseteq Y\}|$. For a given *minimum support threshold* min_sup, an itemset X is a *frequent pattern* if $sup(X) \geq min\_sup$. Given a transaction database and a minimum support threshold, the problem of *frequent pattern mining* [4] is to find the *complete set* of frequent patterns.

For example, consider the transaction database TDB in Figure 1. For the sake of simplicity, we write an itemset as a string of items. For example, itemset $\{a, c, d\}$ is written as *acd*. Let the support threshold $min\_sup = 2$. Since *abcd* is contained in transactions 20, 30 and 40, $sup(abcd) = 3$ and *abcd* is a frequent pattern.

Frequent pattern mining is fundamental for many data mining tasks, such as mining association rules [5], correlations [10], causality [29], sequential patterns [6], episodes [22], partial periodicity [16],

tid	itemset
10	*bcde*
20	*abcd*
30	*abcdf*
40	*abcde*
50	*def*

Figure 1: A transaction database.



{}
a b c d e f
ab ac ad ae af bc bd be bf cd ...
abc abd abe abf acd ace acf ...
...
abcde abcdf abcef abdef acdef bcdef
abcdef
(a) the itemset lattice
(b) a set enumeration tree

Figure 2: An itemset lattice and a set enumeration tree.

iceberg-cube computation [8], associative classification [19], and subspace clustering [3]. It is also important in many applications, such as market analysis and network intrusion detection.

Frequent patterns have the well-known monotonic *Apriori property* [4]: if X is frequent, then every nonempty subset of X is also frequent. For an itemset X, $|X|$ is called the *length* of X. According to the Apriori property, a long frequent pattern of length n leads to $(2^n - 2)$ shorter non-empty frequent patterns. For example, in Figure 1, if $min\_sup = 2$, *abcd* is a frequent pattern. All subsets of *abcd* including a, b, c, d, ab, $\ldots$, bcd are also frequent patterns.

To avoid mining all frequent patterns, we can mine only those max-patterns [7]. An itemset X is a *maximal frequent pattern* or a *max-pattern* for short if X is frequent and every proper superset of X is infrequent. In Figure 1, when $min\_sup = 2$, the max-patterns are *abcd*, *bcde* and *df*. The problem of *mining maximal frequent patterns* (or *mining max-patterns* for short) is to find the *complete set* of max-patterns.

Mining max-patterns efficiently is important in both theory and applications of frequent pattern mining. On the theoretical side, the max-patterns serve as the border between the frequent patterns and the infrequent ones. With the set of max-patterns, whether an itemset is frequent or not can be determined quickly using the Apriori property. On the application side, max-patterns are used in a few interesting and challenging data mining tasks. For example, using max-patterns, we can find emerging patterns [12] which are patterns frequent in the positive samples and infrequent in the negative samples. Emerging patterns can be used to construct effective classifiers [18]. As another example, using max-patterns with respect to a series of support thresholds, we can summarize and approximate the support information of all frequent patterns [26].

More broadly, mining max-patterns is also related to many data mining problems, including mining generators [20], mining borderline description [13], mining maximal sequential patterns [21], web log mining [28], condensed representations of constrained frequent patterns [9], and summarizing association rules [23].

The fundamental challenge of mining max-patterns is how to search a large space of itemsets and identify max-patterns. Most of the existing methods search an enumeration tree of itemsets in a depth-first manner. The search is often arranged according to some heuristics such as the frequencies of items. [2] provides a good survey.

One important and interesting issue overlooked in the previous studies is how the max-patterns already found can help to plan the search of new max-patterns. In this paper, we develop a novel pattern-aware approach which dynamically schedules the search based on the max-patterns already found. A distinct advantage is that many branches in the dynamic scheduled search space can be pruned sharply. We also present efficient techniques to implement the new approach. As indicated by a systematic empirical study using the benchmark data sets, our new approach outperforms the currently fastest max-pattern mining algorithms FPMax* [15] and LCM2 [30] in a clear margin.

2 Search Space and Search Strategies

Due to the Apriori property, only frequent items can appear in a max-pattern. Thus, the search space of max-pattern mining is the lattice of itemsets consisting of only frequent items, which is called the *itemset lattice*. Figure 2(a) shows the itemset lattice of the transaction database in Figure 1, where $I = \{a, b, c, d, e, f\}$, $min\_sup = 2$, and every item is frequent.

Essentially, the itemset lattice can be searched in an either breadth-first or depth-first manner. Consider the transaction database TDB in Figure 1. In a breadth-first search, we start with finding the frequent items, i.e., a, b, c, d, e and f. Then, we combine the frequent items to generate length-2 itemsets, i.e., ab, ac, $\ldots$, ef. The supports of those length-2 candidates are counted, and the length-2 frequent patterns are found. A length-3 pattern X is generated as a candidate only if every length-2 subset of X is frequent. For example, abc is generated as a length-3 candidate since ab, ac and bc are frequent, while def should not be generated as a length-3 candidate since ef is infrequent. The search continues until all candidates of the current iteration are infrequent, or no longer candidates can be generated.

A few methods such as MaxMiner [7] search an itemset lattice in a breadth-first manner. As indicated by some previous studies such as [2], the breadth-first search methods may have to search many patterns that are not maximal or even infrequent.

To reduce the number of patterns searched, some recently developed methods such as Depth-Project [1], Mafia [11], GenMax [14], FPMax* [15] and LCM2 [30] conduct depth-first searches. A global order called the *enumeration order* on all frequent items can be used to enumerate all itemsets systematically in a set enumeration tree [27]. Figure 2(b) shows a set enumeration tree of the lattice in Figure 2(a) where the lexicographic order of items is used as the enumeration order. In the subtree of a, we search for patterns having item a. In the subtree of b, we search for patterns having item b but no item a. The search space of other subtrees can be specified similarly.

Depth-first searches can be implemented efficiently using projected databases. To search patterns in the subtree of a, we only need to check the transactions containing a, which is called the *a-projected database*. Similarly, to search patterns in the subtree of ab, we only need to check the ab-projected database, which is a subset of the a-projected database. Since ab is a child of a in the set-enumeration tree, the depth-first search takes a divide-and-conquer strategy.

A critical pruning technique called *head-and-tail pruning* was firstly proposed in MaxMiner, and was adopted by DepthProject, Mafia, GenMax and FPMax*. Consider Figure 1 again and let $min\_sup = 2$. Suppose we use the lexicographic order of items in a depth-first search of max-patterns. The a-projected database consists of transactions 20, 30 and 40. Items b, c and d are frequent in the a-projected database and form the *tail* of a, denoted by $Tail(a) = bcd$. According to the Apriori property, any pattern X containing a can have only items from $Tail(a)$ or a itself, i.e., $X \subseteq a \cup Tail(a) = abcd$. Before we unfold the subtree of a, we can first check $sup(abcd)$. Since $abcd$ is frequent and no other max-patterns found later will contain a (due to the divide-and-conquer partitioning in the set-enumeration tree), $abcd$ is a max-pattern. Any frequent pattern in the subtree of a must be a subset of $abcd$ and thus cannot be a max-pattern. We do not need to search the subtree. Similarly, we can find max-pattern $bcde$ from the subtree of b. Now, let us consider c. $Tail(c) = de$ which means any pattern containing c but no a or b must be a subset of $c \cup Tail(c) = cde$. Since cde is a subset of $bcde$, a max-pattern found before, the subtree of c does contain any max-pattern and thus can be pruned immediately.

In the head-and-tail pruning, finding long max-patterns early may prune more subtrees. A heuristic called *dynamically ordering of frequent items* was firstly proposed in MaxMiner, and was adopted by DepthProject, Mafia, GenMax and FPMax*. When we search the subtree of itemset X, we find the set of items that are frequent in the X-projected database. We sort those frequent items in the support ascending order to construct the subtree of X. The rationale is that a set enumeration tree constructed using such an order may have a small left subtree, and may lead to max-patterns early.

The previous studies such as [2] suggest that depth-first searches often have a better performance than breadth-first searches in mining max-patterns.

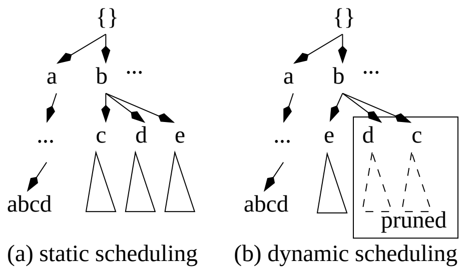


Figure 3: The pruning effect of pattern-aware dynamic search.

3 Pattern-Aware Dynamic Search

Consider TDB in Figure 1 again. From the a-projected database, we can find max-pattern *abcd*. Now, let us consider how to search the subtree of b. $Tail(b) = cde$. Since $b \cup Tail(b) \not\subseteq abcd$, we need to search the subtree of b.

If we use the order of c-d-e to enumerate patterns containing b but no a, as shown in Figure 3(a), then, we need to search all the three subtrees of the children of b, namely *bc*, *bd* and *be*.

Given *abcd* is a max-pattern found before, one critical observation here is that e is the only item in $b \cup Tail(b)$ but not in *abcd*. Any pattern in the subtree of b not containing item e is a subset of max-pattern *abcd* and thus cannot be a max-pattern. In other words, in the subtree, we only need to search the patterns containing e. Thus, in the subtree of b, if we sort the items such that e precedes c and d, as shown in Figure 3(b), we only need to search the subtree of e, and the subtrees of c and d can be pruned immediately. In fact, the ordering between c and d does not matter.

The essential idea of pattern-aware dynamic search is simple. When a subtree is searched, based on the max-patterns found before, we construct the subtree in a way that the potential max-patterns are scheduled into some branches that have to be searched, and the patterns that are subsets of max-patterns found before are organized into branches that can be pruned.

Let us generalize the idea technically. Suppose we want to search the subtree of an itemset X. Let $Y \supset X$ be a max-pattern found before. Then, we can schedule the search of the subtree of X as follows. We partition $Tail(X)$ into two subsets: $T_1 = Tail(X) - Y$ is the set of items that do not appear in Y; and $T_2 = Tail(X) \cap Y$ is the set of items that appear in Y. Any max-pattern in the subtree of X must contain at least one item from T_1. Thus, we order the items such that the items in T_1 precede the items in T_2.

Using this order, we only need to search the children of X that are in T_1. The children of X in T_2 and their subtrees can be pruned immediately. The above process is called the *pattern-aware dynamic search* (*PADS* for short), and an order where items in T_1 precede items in T_2 is called a

PADS order with respect to Y. Max-pattern Y is called the *key pattern* of the search scheduling.

We prove the correctness of the above scheduling.

Theorem 1 (Correctness) *Let X be a frequent pattern, and Y be a max-pattern such that $X \subset Y$. If a PADS order with respect to Y is used to construct the set enumeration subtree of X, then for any item $z \in Tail(X) \cap Y$ and any pattern Z in the subtree of $X \cup \{z\}$, $Z \subset Y$.*

Proof. As discussed before, $Tail(X \cup \{z\}) \subset Tail(X)$. Since a PADS order with respect to Y is used, z is behind all items in $Tail(X) - Y$ in the order. That is, $Tail(X \cup \{z\}) \subset Y$. Since $z \in Tail(X) \cap Y$ and $X \subset Y$, we have $Z \subseteq (X \cup \{z\} \cup Tail(X \cup \{z\})) \subset Y$. ■

The pattern-aware dynamic search technique is different from the technique of dynamically ordering frequent items developed in the previous studies, which uses the item frequency ascending order to construct a set enumeration subtree. Dynamically ordering frequent items is a heuristic method. Due to the correlations among frequent items, there exist counter examples where sorting frequent items in support ascending order does not help pruning. In contrast, the effect of pattern-aware dynamic search is determined once the key pattern is chosen. To search the subtree of a pattern X, once there exists at least one key pattern $Y \supset X$ found before, a PADS order based on Y can be used to prune some children of X by pattern-aware dynamic search. It is not heuristic.

4 Choosing a Good PADS Order

To the best our knowledge, LCM2 [30] is the only existing method adopting a similar idea in mining max-patterns. What is the critical difference between our method and LCM2?

For a pattern X, if there are more than one pattern Y such that $Y \supset X$, then each pattern can serve as a key pattern, and thus multiple PADS orders are feasible. Now, the problem becomes how to select a good PADS order.

In LCM2, an arbitrary item e in the tail of Y is picked, and the max-patterns containing $Y \cup \{e\}$ are mined. Then, the longest max-pattern containing $Y \cup \{e\}$ is chosen as the key pattern, and the PADS order is determined accordingly. However, the method may not lead to good performance all the time.

First, issuing a sub-routine to find all max-patterns containing $Y \cup \{e\}$ may lead to searching a large part of the subtree of $Y \cup \{e\}$. Those max-patterns are not necessarily good since e is chosen arbitrarily.

Instead of searching many new max-patterns containing $Y \cup \{e\}$, *PADS* reuses the max-patterns already found as much as possible to find a good key pattern. Therefore, we avoid the cost of searching many new max-patterns in order to scheduling the future search.

Second, the longest max-pattern may not be always good. For example, suppose the current pattern $Y = fgh$, $tail(Y) = ijk$, and item i is chosen. Furthermore, suppose the longest max-pattern found containing $Y \cup \{i\} = fghi$ is $X_1 = abcdfghi$. It in fact does not provide any pruning power in the scheduling. Suppose another max-pattern $X_2 = efghik$ is found before. Then, X_2 provides a good pruning power in the scheduling: we only need to search the $Y \cup \{j\}$ subtree.

Instead of choosing key patterns based on length, *PADS* measures the pruning powers of the max-patterns already found, and selects key patterns accordingly.

As analyzed, the effect of the pattern-aware dynamic search technique depends on the choice of key patterns. In this section, we discuss how to choose a good key pattern.

Let us consider choosing a key pattern for an itemset X. If $Y \supset X$ is chosen, as indicated by Theorem 1, all children of X in $Tail(X) \cap Y$ can be pruned. Therefore, the more items in $Tail(X)$ appear in the key pattern, the more children can be pruned. Thus, we can choose a max-pattern $Y \supset X$ as the key pattern such that Y has the largest overlap with $Tail(X)$. That is,

$$Y = \arg\max_{\text{max-pattern } Z \supset X}\{|Z \cap Tail(X)|\}$$

Please note that Y contains at least one item that is not in $Tail(X)$. Otherwise, $X \cup Tail(X)$ is a subset of Y, and thus X is pruned by the head and tail pruning technique. In the example shown in Figure 3(b), pattern *abcd* is a perfect choice for b since we only need to search one child of b.

We choose key patterns for itemset X in two steps.

4.1 Step 1

In the first step, we check all max-patterns Y_1 found before that are supersets of X and measure $|Y_1 \cap Tail(X)|$. The max-pattern with the most overlap with $Tail(X)$ is chosen as the candidate.

This step can be implemented as a byproduct of the head-and-tail pruning. For each itemset X, to apply the head-and-tail pruning, we have to check $X \cup Tail(X)$ against the max-patterns found so far. We also collect the information of $|Y_1 \cap Tail(X)|$ at the same time. Thus, the cost of computing the candidate in this step is very little.

There can be many (millions or more) max-patterns found so far in a large database. To speed up checking whether X is a subset of some max-patterns found before, we adopt the *progressive focusing search strategy* developed in GenMax. When we search an itemset X and its subtree, any patterns found in the subtree must be a superset of X. Thus, we can maintain the set of max-patterns found so far that are supersets of X. Any patterns found in the subtree of X only need to be checked against those max-patterns.

The technique can be applied recursively. For itemset $X \cup \{y\}$ that is a child of X in the set enumeration tree, the max-patterns containing $X \cup \{y\}$ is a subset of those containing X. Thus, the maintenance of the matching max-patterns is progressive.

4.2 Step 2

In some situations, max-patterns found so far may not have heavy overlaps with the tail of X. Thus, as the second step, we also find in the projected database of X's parent one max-pattern Y such that $Y_2 \supset X$ and $Y_2 \subset X \cup Tail(X)$. This can be done quickly as follows. According to the order used by the parent of X, items in $Tail(X)$ can be ordered into a list, say $x_1, \ldots, x_n$. Since x_1 is frequent in the X-projected database, $X \cup \{x_1\}$ must be frequent. We first assign $Y_2 = X \cup \{x_1\}$ and check the supports of $Y_2 \cup \{x_2\}$, $Y_2 \cup \{x_3\}$, ..., $Y_2 \cup \{x_n\}$. If none of them is frequent, then Y_2 is the candidate key pattern. Otherwise, let i_1 be the smallest index number such that $Y_2 \cup \{x_{i_1}\}$ is frequent. Then, we update Y_2 to $Y_2 \cup \{x_{i_1}\}$. We recursively use $x_{i_1+1}, x_{i_1+2}, \ldots, x_n$ to expand Y_2 until it cannot be expanded longer. It is easy to see that the pattern Y_2 found as such is a max-pattern if it is not a subset of a max-pattern found before.

By the second step, we can find at least one max-pattern that can be used as a key pattern. We compare the two key pattern candidates found from the two steps, and pick the one Y having the better pruning power as the key pattern. The PADS order is made accordingly.

In implementation, we use FP-trees [17] as the core data structure to store transactions and projected databases. We also integrate the advantages in the existing methods. Particularly, we adopt the *pattern expansion technique* which was firstly proposed in CLOSET [25] and CHARM [32] in the context of frequent closed itemset mining, and later used by Mafia and GenMax in max-pattern mining. Consider the situation where every transaction containing itemset X also contains item $y \in Tail(X)$. Then, it is impossible that a max-pattern contains X but does not contain y. Therefore, we do not need to search any subtree of X where y does not appear. In other words, instead of searching the subtree of X, we can directly search the subtree of $X \cup \{y\}$.

The algorithm *PADS* (for pattern-aware dynamic search) is summarized in Figure 4. Moreover, we make the source code and the executable code (on both Windows and Linux platforms) publicly available at `http://www.cs.sfu.ca/~jpei/Software/PADS.zip`.

Complexity Analysis

As indicated in [31], the problem of mining max-patterns is NP-hard. Therefore, all max-pattern mining algorithms developed so far unfortunately have the exponential complexity.

Our *PADS* method shares the same depth-first search framework with the state-of-the-art, depth-first search methods such as FPMax* and LCM2. To analytically understand the efficiency of the

```
Input: a transaction database $TDB$ and support threshold $min\_sup$;
Output: the set of max-patterns;
Method:
1:  find $I$, the set of frequent items in $TDB$;
2:  CALL $PADS(TDB, \emptyset, I)$;

 Function $PADS(PDB, X, T)$ // $PDB$ is the $X$-projected database, $T$ is the tail of $X$
11: let $Z = \{z \in T | sup(z) = |PDB|\}, X = X \cup Z, T = T - Z$;
    // pattern expansion
12: FOR EACH item $x$ in $T$ DO
13:    $X' = X \cup \{x\}$;
14:    let $PDB_x$ be the $X'$-projected database;
15:    $T'$ = the set of frequent items in $PDB_x$;
16:    IF $(X' \cup T')$ is a subset of some max-pattern found before THEN RETURN;
17:    let $Y_1$ be candidate key pattern as the max-pattern with the largest overlap with $T'$
       obtained as the byproduct of the subpattern checking; // Section 4.1
18:    let $Y_2$ be the candidate key pattern obtained from the projected database $PDB$;
       // Section 4.2
19:    IF $Y = Y_2$ and $Y_2$ is a max-pattern THEN output $Y_2$;
20:    let $Y$ be the better key pattern between $Y_1$ and $Y_2$;
21:    make a PADS order on $T'$ according to $Y$;
22:    CALL $PADS(PDB_x, X', T')$;
    END FOR
    RETURN
```

Figure 4: The *PADS* algorithm.

PADS method, the critical issue is to analyze the cost of implementing the pattern-aware dynamic search. Particularly, the cost of finding key patterns in *PADS* is important.

First of all, let us consider the complexity of finding the first max-pattern. Algorithm *PADS* works as any depth-first search max-pattern mining algorithm. It starts with the first frequent item x_1 in the alphabetical order[1], and sets pattern $Y = x_1$. Recursively, a projected database TDB_Y is formed and the frequent items in TDB_Y are found. The first frequent item in the alphabetical order in TDB_Y, say x, is used to expand Y to a longer pattern $Y \cup \{x\}$. The recursion continues until no frequent item can be found in the projected database.

[1]In fact, any total order works here. For the sake of simplicity, we use alphabetical order in our discussion.

Clearly, we have the following result.

Lemma 1 *The time complexity of finding the first max-pattern is $O(|TDB| \cdot l)$ where l is the length of the longest transaction in TDB.*

Proof. Trivially, a projected database can be formed and the frequent items in the projected database can be found in time $O(|TDB|)$. There are at most l recursion steps is needed to find the first max-pattern, since any frequent pattern cannot be longer than the length of the longest transaction. Thus, we have the lemma. ■

In implementation, PADS adopts pseudo-projection [24] to find the first max pattern. In pseudo projection, no physical projected databases are constructed. Instead, PADS only manipulates pointers to construct "virtual" projected database.

As analyzed before, for an itemset X, algorithm *PADS* chooses a key pattern for X in two steps. In the first step, *PADS* finds a max-pattern Y_1 found before which maximizes the overlap between Y_1 and $Tail(X)$. Clearly, the complexity of this step is linear with respect to the number of max-patterns found so far that contain X.

In the second step, *PADS* finds a max-pattern containing X by considering the items in $Tail(X)$. Therefore, the complexity of this step is linear with respect to $|Tail(X)|$. Moreover, since a max-pattern cannot be longer than l, the length of the *min_sup*-th longest transaction in the X-projected database, the cost of this step is also linear with respect to l. In the worst case where $min\_sup = 1$, l is the length of the longest transaction in the X-projected database. Last, the cost in this step is linear with respect to the number of transactions in the X-projected database, since *PADS* needs to scan the database iteratively to count the support of items in $Tail(X)$. In summary, we have the following claim about the cost in the second step.

Lemma 2 *For an itemset X, the cost of Line 8 in Figure 4 is $O(|TDB_X| \cdot \min\{|Tail(X)|, l\})$, where l is the length of the longest transaction in the X-projected database.* ■

Taking both the cost of the two steps in finding key patterns together, we have the following result about the cost of finding a key pattern.

Theorem 2 *For an itemset X, the cost of finding a key pattern for X in* PADS *is $O(m + |TDB_X| \cdot \min\{|Tail(X)|, l\})$, where m is the number of max-patterns containing X and l is the length of the longest transaction in the X-projected database.* ■

How can we compare the cost of finding key patterns against the benefit of pruning sub-trees in the set-enumeration search space using the pattern-aware dynamic search? One important observation

Data set	# tuples	# items	avg trans len
Chess	$3,196$	76	37
Mushroom	$8,124$	120	23
Pumsb*	$49,046$	$2,088$	50
Pumsb	$49,046$	$2,113$	74
Connect	$67,557$	150	43

Table 1: Characteristics of benchmark data sets.

is that the cost of determining whether a frequent pattern X is maximal is $O(m)$, where m is the number of max-patterns containing X. Therefore, the cost of finding a key pattern is mainly the cost of step 2, which is of complexity $O(|TDB_X| \cdot \min\{|Tail(X)|, l\})$. Following with the results in [31], the cost of finding all max-patterns in a sub-tree rooted at X in the set-enumeration search space is of complexity $O(2^{|Tail(X)|})$. Therefore, once a pruning case happens in *PADS*, we save the search cost of $O(2^{|Tail(X)|})$ using a key pattern searching cost $O(|TDB_X| \cdot \min\{|Tail(X)|, l\})$.

Both *PADS* and LCM2 use pattern-aware dynamic search. Then, what is the difference between their efficiency? For an itemset X, LCM2 chooses an arbitrary item $x \in Tail(X)$ and use the longest max-pattern containing $X \cup \{x\}$ as the key pattern. The cost of finding such a key pattern is of complexity $O(2^{|Tail(X \cup \{x\})|})$, which is much higher that the cost of key pattern finding in *PADS*. However, the effectiveness of the key pattern chosen by LCM2 may not be always better than that of *PADS* since *PADS* considers all max-patterns found so far (step 1) and also one max-pattern of good coverage in $Tail(X)$ (step 2). Our experimental results clearly show that *PADS* outperforms LCM2 in both the number of patterns checked and the number of projected databases generated.

5 Empirical Evaluation

We conducted an extensive performance study to evaluate the effectiveness of the pattern-aware dynamic search and the efficiency of our *PADS* algorithm. Here we report the experimental results on five real data sets. Those five real data sets were prepared by Roberto Bayardo from the UCI datasets and PUMSB. They have been used extensively in the previous studies as the benchmark data sets. Some characteristics of the five data sets are shown in Table 1. We downloaded the data sets from `http://fimi.cs.helsinki.fi/data/`.

All the experiments were conducted on a PC computer running the Microsoft Windows XP SP2 Professional Edition operating system, with a 3.0 GHz Pentium 4 CPU, 1.0 GB main memory, and a 160 GB hard disk. The programs were implemented in C/C++ using Microsoft Visual Studio. NET

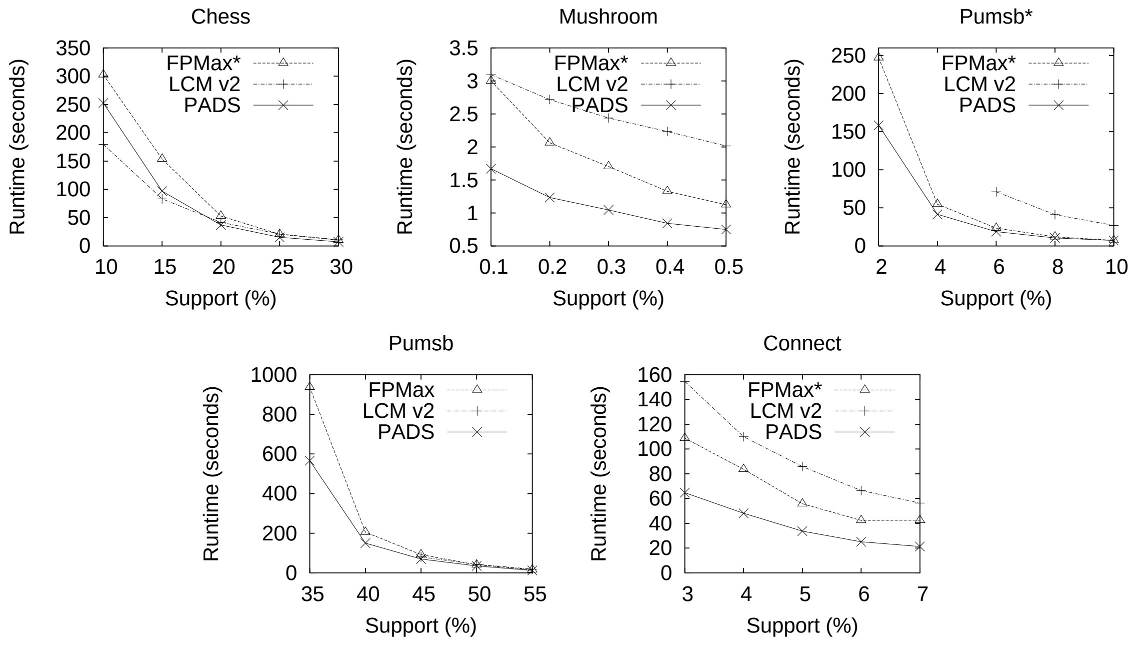


Figure 5: The runtime comparison between *PADS* and FPMax* on the five benchmark data sets.

2003.

We compare our method with FPMax* and LCM2, the currently fastest max-pattern mining methods according to the extensive empirical study reported in the Frequent Itemset Mining Implementations Repository website (http://fimi.cs.helsinki.fi/). We used the code of the two algorithms published by the authors.

It should be mentioned that LCM2 has execution problems under some circumstances. On the Pumsb data set with min_sup lower than 40%, on the Pumsb* data set with min_sup lower than 6%, and on the Connect data set with min_sup lower than 0.2%, LCM2 gives segmentation faults and cannot finish properly. Therefore parts of its curves are missing.

Figure 5 shows the runtime comparison among the three algorithms on the five data sets. In the figures, a support threshold is presented as a percentage with respect to the total number of transactions in the data set, i.e., $\frac{min\_sup}{|D|}$ where D is the data set in question.

Figure 5 clearly shows that *PADS* outperforms FPMax* on all data sets. The lower the support threshold, the larger the difference in runtime. With a smaller support threshold, more patterns and longer patterns are qualified as frequent patterns. This trend suggests that *PADS* is more scalable than FPMax* on mining a large number of long patterns. When the support threshold is low, the difference in runtime between the two methods can be more than 60%.

Most of the time, *PADS* outperforms LCM2 clearly, especially on the Mushroom and the Connect data sets. The only circumstance where LCM2 outperforms *PADS* is on the Chess data set with $min\_sup \leq 15\%$. The reason is that the number of max-patterns is large (more than 1 million) but the database size is very small (only $3,196$ tuples). The advantage of selecting a good key pattern is not clear in this situation.

What are the major reasons that *PADS* outperforms FPMax* and LCM2? The major cost of max-pattern mining in depth-first manner comes from two aspects: generating projected databases and checking whether a pattern is a subset of some max-patterns found before.

In Figure 6, we compare the three methods in terms of the number of projected databases generated on the five data sets. *PADS* generates much (about 80%) less projected databases than FPMax*. LCM2 generates the largest number of projected databases. This clearly shows the power of pattern-aware dynamic search in *PADS*. Many subtrees can be pruned by scheduling using good key patterns carefully chosen by our method.

In Figure 7, we compare the three methods in terms of the number of patterns that are checked against the max-patterns found before. *PADS* also conducts less subpattern checking than FPMax* and LCM2. The reason is that the pattern-aware dynamic search prunes many subtrees. Some patterns in those subtrees that are checked in FPMax* and LCM2 do not need to be checked by *PADS*. The savings in generating projected databases and checking subpatterns explain the advantage of *PADS* in performance.

Last, Figure 8 compares the memory usage of the three methods. Both *PADS* and FPMax* use FP-trees as the major data structure. Thus, their memory usage is very similar. Their memory usage increases as the support threshold decreases, since they store the max-patterns already found in memory, and the number of such patterns increases as the support threshold decreases. On the other hand, LCM2 stores max-patterns on disk, and uses an array in main memory to store the database and the projected databases. Thus, its memory usage is insensitive with respect to support thresholds. In large data sets such as Mushroom and Connect, *PADS* and FPMax* use less memory than LCM2. In small data sets, LCM2 consumes a smaller amount of main memory than the other two methods.

6 Conclusions

Max-pattern mining is important in both theory and applications of frequent pattern mining. In this paper, we developed a novel pattern-aware dynamic search method for fast max-pattern mining. The major idea is to schedule the depth-first search according to the max-patterns found so far, and prune the search space systematically. We present efficient methods to implement pattern-aware

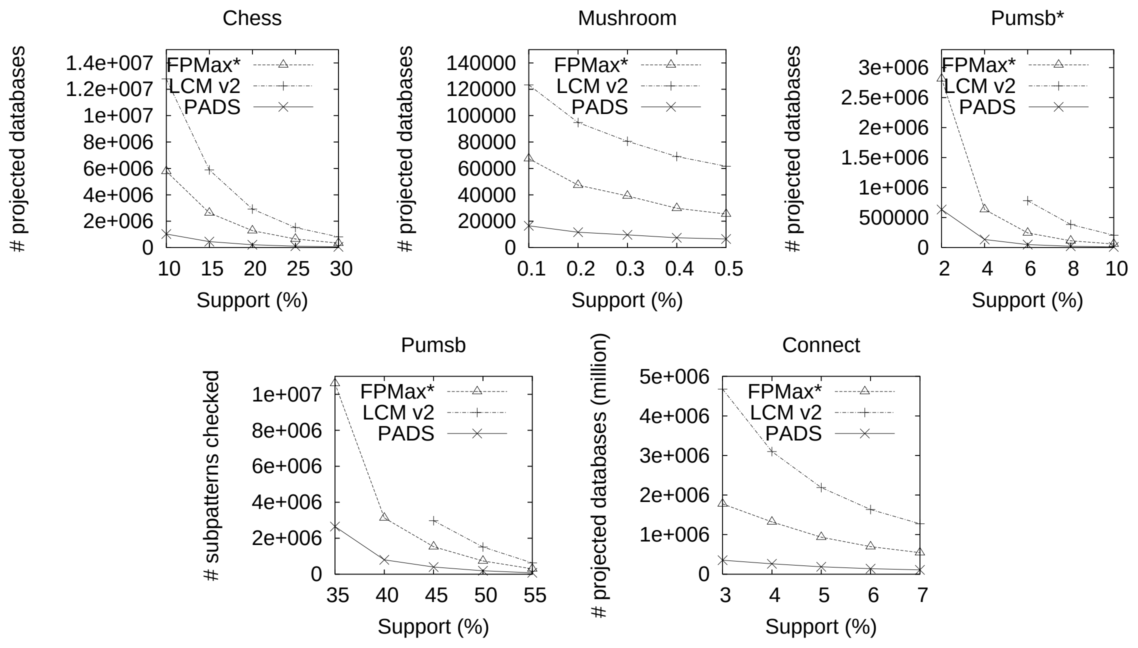


Figure 6: Number of projected databases generated.

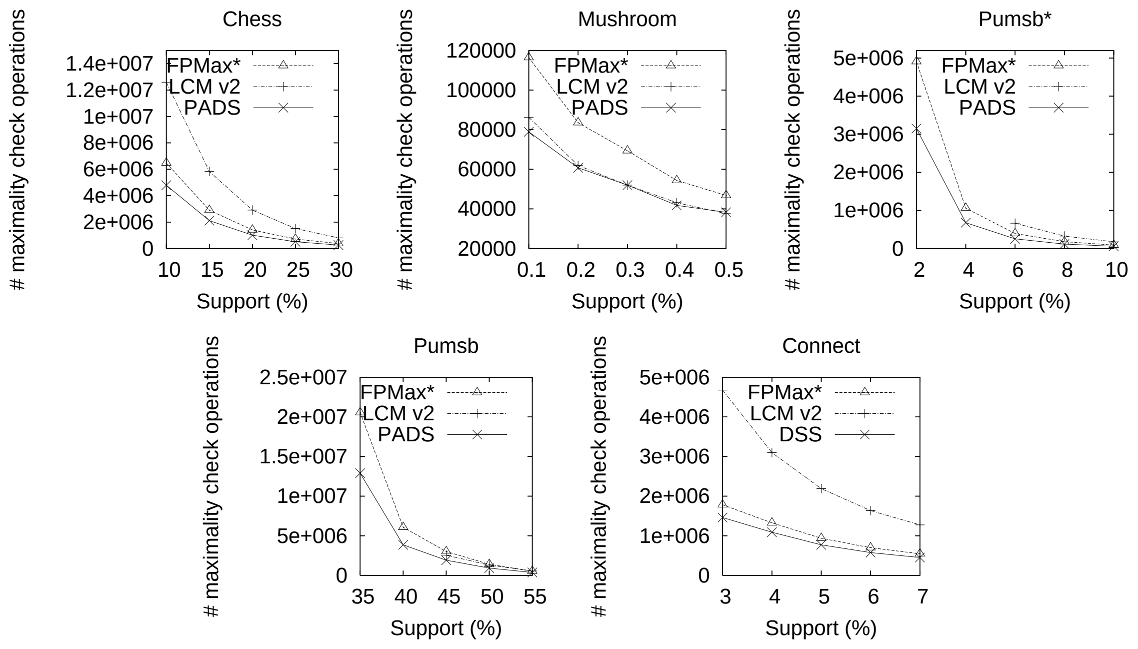


Figure 7: Number of patterns checked against max-patterns.

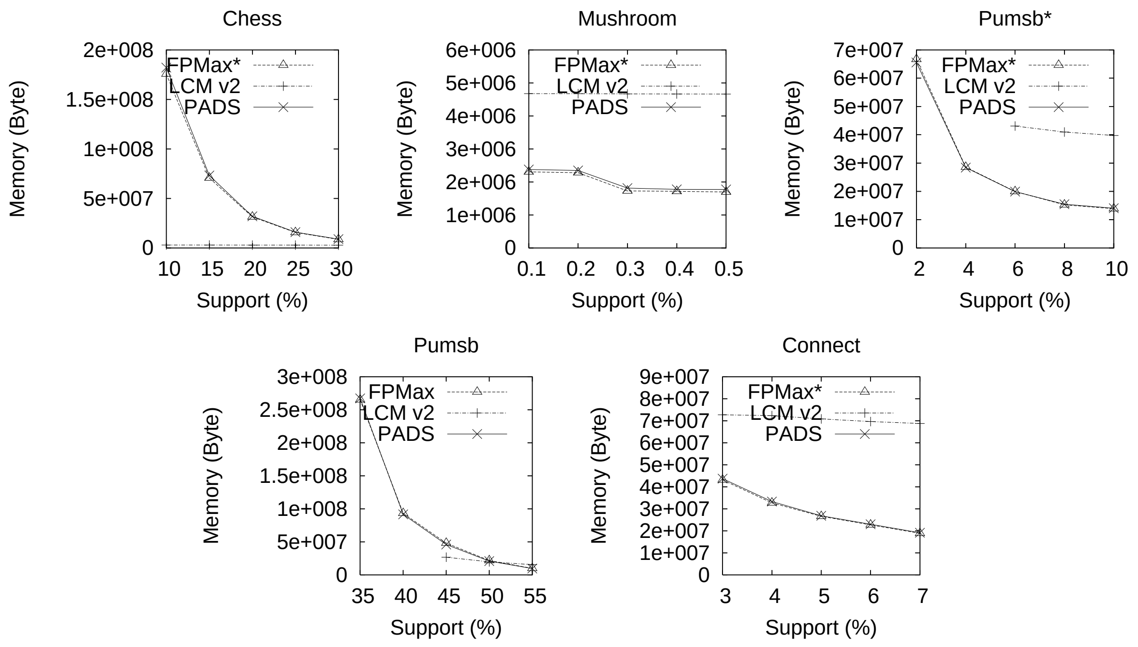


Figure 8: Memory usage.

dynamic search. An empirical evaluation using the benchmark real data sets clearly shows that our method outperforms the currently fastest max-pattern mining algorithms FPMax* and LCM2 in a clear margin.

As future work, it is interesting to explore how the pattern-aware dynamic search method can be extended to other frequent pattern mining tasks, such as mining frequent closed itemsets, max- and closed sequential patterns, and max- and closed graph patterns.

References

[1] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 108–118, Boston, Massachusetts, United States, 2000. ACM Press.

[2] Charu C. Aggarwal. Towards long pattern generation in dense databases. *SIGKDD Explor. Newsl.*, 3(1):20–26, 2001.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 94–105, Seattle, WA, June 1998.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93)*, pages 207–216, Washington, DC, May 1993.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.

[6] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[7] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 85–93, Seattle, WA, June 1998.

[8] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 359–370, Philadelphia, PA, June 1999.

[9] Francesco Bonchi and Claudio Lucchese. On condensed representations of constrained frequent patterns. *Knowl. Inf. Syst.*, 9(2):180–201, 2006.

[10] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, pages 265–276, Tucson, Arizona, May 1997.

[11] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 443–452, Heidelberg, Germany, April 2001.

[12] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, pages 43–52, San Diego, CA, Aug. 1999.

[13] Guozhu Dong and Jinyan Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowl. Inf. Syst.*, 8(2):178–202, 2005.

[14] Karam Gouda and Mohammed Javeed Zaki. Efficiently mining maximal frequent itemsets. In *ICDM*, pages 163–170, 2001.

[15] Gosta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, 2005.

[16] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, pages 106–115, Sydney, Australia, April 1999.

[17] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

[18] Jinyan Li, Guozhu Dong, Kotagiri Ramamohanarao, and Limsoon Wong. Deeps: A new instance-based lazy discovery and classification system. *Mach. Learn.*, 54(2):99–124, 2004.

[19] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, New York, NY, Aug. 1998.

[20] Guimei Liu, Jinyan Li, and Limsoon Wong. A new concise representation of frequent itemsets using generators and a positive border. *Knowl. Inf. Syst. (accepted).*

[21] Congnan Luo and Soon M. Chung. A scalable algorithm for mining maximal frequent sequences using a sample. *Knowl. Inf. Syst. (accepted).*

[22] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[23] Carlos Ordonez, Norberto Ezquerra, and Cesar A. Santana. Constraining and summarizing association rules in medical data. *Knowl. Inf. Syst.*, 9(3):1–2, 2006.

[24] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-Mine: Hyper-structure mining of frequent patterns in large databases. In *Proc. 2001 Int. Conf. Data Mining (ICDM'01)*, pages 441–448, San Jose, CA, Nov. 2001.

[25] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)*, pages 11–20, Dallas, TX, May 2000.

[26] Jian Pei, Guozhu Dong, Wei Zou, and Jiawei Han. Mining condensed frequent-pattern bases. *Knowl. Inf. Syst.*, 6(5):570–594, 2004.

[27] R. Rymon. Search through systematic set enumeration. In *Proc. 1992 Int. Conf. Principle of Knowledge Representation and Reasoning (KR'92)*, pages 539–550, Cambridge, MA, 1992.

[28] Mehmet Sayal and Peter Scheuermann. Distributed web log mining using maximal large item sets. *Knowl. Inf. Syst.*, 3(4):389–404, 2001.

[29] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 594–605, New York, NY, Aug. 1998.

[30] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, Brighton, UK, November 2004.

[31] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'04)*. ACM Press, 2004.

[32] M. J. Zaki and C. J. Hsiao. Charm: An efficient algorithm for closed association rule mining. In *Technical Report 99-10*, Computer Science, Rensselaer Polytechnic Institute, 1999.