Mining Colossal Frequent Patterns by Core Pattern Fusion*

Feida Zhu† Xifeng Yan‡ Jiawei Han† Philip S. Yu‡ Hong Cheng†

†University of Illinois at Urbana-Champaign {feidazhu, hanj, hcheng3}@cs.uiuc.edu
‡IBM T. J. Watson Research Center {xifengyan,psyu}@us.ibm.com

Abstract

Extensive research for frequent-pattern mining in the past decade has brought forth a number of pattern mining algorithms that are both effective and efficient. However, the existing frequent-pattern mining algorithms encounter challenges at mining rather large patterns, called colossal *frequent patterns, in the presence of an explosive number of frequent patterns. Colossal patterns are critical to many applications, especially in domains like bioinformatics. In this study, we investigate a novel mining approach called* Pattern-Fusion *to efficiently find a good approximation to the colossal patterns. With Pattern-Fusion, a colossal pattern is discovered by fusing its small* core patterns *in one step, whereas the incremental pattern-growth mining strategies, such as those adopted in Apriori and FP-growth, have to examine a large number of mid-sized ones. This property distinguishes Pattern-Fusion from all the existing frequent pattern mining approaches and draws a new mining methodology. Our empirical studies show that, in cases where current mining algorithms cannot proceed, Pattern-Fusion is able to mine a result set which is a close enough approximation to the complete set of the colossal patterns, under a quality evaluation model proposed in this paper.*

1 Introduction

Frequent pattern mining is one of the most important data mining problems that has been well recognized over the past decade. A pattern is frequent if and only if it occurs in at least σ fraction of a dataset, where σ is user-defined. It is essential to a broad range of applications including association rule mining [2, 14], time-related process and scientific sequence data analysis, bioinfomatics, classification, indexing and clustering. Intense research on this topic has produced a series of mining algorithms for finding frequent patterns in large databases of itemsets, sequences and graphs [16, 22, 11]. For many applications, these algorithms have proved to be effective. Efficient open source implementations were also available over years. For example, FPClose [8] and LCM2 [18] (an improved version of MaxMiner [3]) published in 2003 and 2004 Frequent Itemset Mining Implementations Workshop (FIMI) can report the complete set of frequent itemsets in a few seconds for reasonably large data sets.

However, the frequent pattern mining problem, even for frequent itemset mining, has not been completely solved for the following reason: According to frequent pattern definition, any subset of a frequent itemset is frequent. This well-known downward closure property leads to an explosive number of frequent patterns. The introduction of closed frequent itemsets [16] and maximal frequent itemsets [9, 3] partially alleviated this redundancy problem. A frequent pattern is *closed* if and only if a super-pattern with the same support does not exist. A frequent pattern is *maximal* if and only if it does not have a frequent super-pattern. Unfortunately, for many real-world mining tasks with increasing importance, such as microarray data analysis in bioinformatics and frequent graph pattern mining, it often turns out that the mining results, even those for closed or maximal frequent patterns, are explosive in size.

It comes with no surprise that this phenomenon should fail all mining algorithms which attempt to report the complete answer set. Take one microarray dataset, ALL [6], for example, which contains 38 transactions each with 866 items. Our experiments show that, when given a low support threshold of 10, FPClose, LCM2 and TFP (top-k) [19] all failed to complete execution.

More importantly, mining tasks in practice usually attach much greater importance to patterns that are larger in pattern size, *e.g.*, longer sequences are usually of more significant meaning than shorter ones in bioinfomatics. We call these large patterns *colossal* patterns, as distinguished from the patterns with large support set. When the complete mining result set is prohibitively large, yet only the colossal ones are of real interest and there are, as in most

* The work was supported in part by the U.S. National Science Foundation NSF IIS-05-13678/06-42771 and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

cases, merely a few of them, it is inefficient to wait forever for the mining algorithm to finish running, when it actually gets "trapped" at those mid-sized patterns. Here is a simple example to illustrate the scenario. Consider a 40×40 square table with each row being the integers from 1 to 40 in increasing order. Remove the integers on the diagonal, and this gives a 40×39 table, which we call $Diag_{40}$. Add to $Diag_{40}$ 20 identical rows, each being the integers 41 to 79 in increasing order, to get a 60×39 table. Take each row as a transaction and set the support threshold at 20. Obviously, it has an exponential number (*i.e.*, $\binom{40}{20}$) of mid-sized closed/maximal frequent patterns of size 20, but only one that is colossal: $\alpha = (41, 42, \ldots, 79)$ of size 39. We checked several fast itemset mining algorithms, including FPClose [8] (the winner of FIMI'03), LCM2 [18] (the winner of FIMI'04). It turned out that none of them can finish within 10 hours. A visualization of the pattern search space is illustrated in Figure 1.

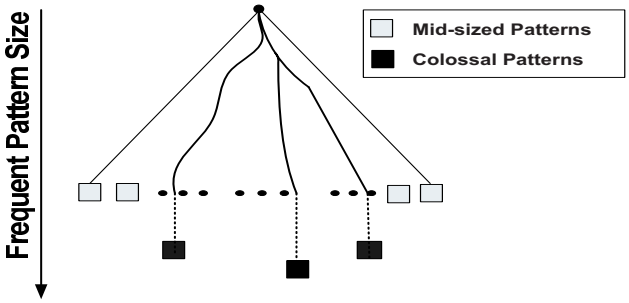

Figure 1. Pattern Search Space

Each node in the search space is a pattern. Nodes at level i is of size i. Node β is a child of node α if and only $\alpha \subset \beta$ and $|\beta| = |\alpha| + 1$. Both breadth-first and depth-first style mining strategies would have to spend exponential time when the number of closed or maximal mid-sized patterns explodes, even though there are only a few colossal patterns.

It should become clear by now that, in these cases, what we need is an efficient computation of a subset of the complete frequent pattern mining result which gives a good approximation to the colossal patterns. The goodness of such an approximation is measured by how well it represents the set of colossal ones among the complete set. Consequently, it motivates us to solve the following problem: ***How to efficiently find a good approximation to the* colossal *frequent patterns?***

There have been some recent work on pattern summarization [21] focusing on post-processing of the complete mining result in order to give a compact answer set. These approaches do not apply for our problem as we intend to avoid the generation of the complete mining set in the first place. A closer examination of the current mining models would expose the insurmountable difficulty posed by this mining challenge: As a result of their inherent mining models in which candidates are examined by implicitly or explicitly traversing a search tree in either a breadth-first or depth-first manner, when the search tree is exponential in size at some level, such exhaustive traversal has to run with an exponential time complexity.

This motivates us to develop a new mining model to attack the problem. Our mining strategy, *Pattern-Fusion*, distinguishes itself from all the existing ones. Pattern-Fusion is able to fuse small frequent patterns into colossal patterns by taking leaps in the pattern search space. It avoids the pitfalls of both breadth-first and depth-first search by applying the following concepts.

1. Pattern-Fusion traverses the tree in a bounded-breadth way. It always pushes down a frontier of a bounded-size candidate pool, *i.e.*, only a fixed number of patterns in the current candidate pool will be used as starting nodes to go downwards in the pattern tree. As such, it avoids the problem of exponential search space.

2. Pattern-Fusion has the capability to identify "shortcuts" whenever possible. The growth of each pattern is not performed with one item addition, but an agglomeration of multiple patterns in the pool. These shortcuts will direct Pattern-Fusion down the search tree much more rapidly toward the colossal patterns.

Figure 2 conceptualizes this mining model.

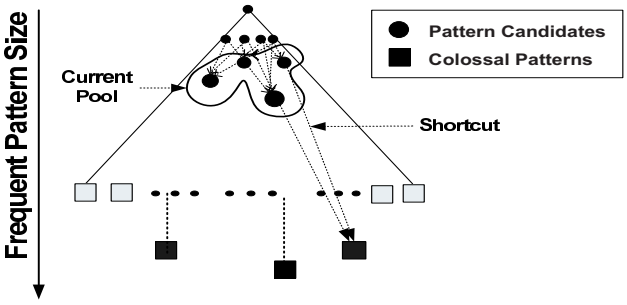

Figure 2. Pattern Tree Traversal

As Pattern-Fusion is designed to give an approximation to the colossal patterns, a quality evaluation model is introduced in this paper to assess the result returned by an approximation algorithm. This could serve as a framework under which other approximation algorithms can be evaluated. Our empirical study shows that Pattern-Fusion is able to efficiently return answers of high quality.

The main contributions of our paper are outlined as follows:

1. We studied the characteristics of colossal frequent itemsets and proposed the concept of *core pattern*. Properties of core patterns that are useful in the mining process are explored. The essential idea exposed in this paper,

can be extended to pattern mining in more complicated data sets, such as sequences and graphs.

2. A new mining model, *Pattern-Fusion*, is introduced, which is different from all existing frequent pattern mining models. Based on the core pattern concept, Pattern-Fusion is the first mining algorithm that generates an approximation set of the colossal patterns *directly in the mining process*.
3. A quality evaluation model is proposed to assess the mining result. This model is able to measure the distance between two arbitrary pattern sets, thus providing a way to measure the goodness of an approximate solution against a complete solution.
4. Empirical studies are conducted on both synthetic and real data sets, demonstrating that (1) Pattern-Fusion gave high quality colossal pattern mining results on data sets that the current mining algorithms can handle; and (2) the method is able to mine colossal patterns out of data sets that no existing mining algorithm would complete in a reasonable amount of time (*e.g.*, 24 hours).

The rest of the paper will be presented as follows: Section 2 reveals the design of Pattern-Fusion based on the concept of core pattern and gives an overview of the mining framework. Section 3 continues to explore the underlying foundation for Pattern-Fusion's ability to mine colossal patterns. The algorithm is elaborated in Section 4. Section 5 proposes the quality evaluation model. Section 6 reports experimental results on various data sets. Related work is discussed in Section 7. We conclude our paper in Section 8.

2 Pattern-Fusion: Design and Overview

2.1 Preliminaries

Let $\mathcal{I}$ be a set of items $\{o_1, o_2, \ldots, o_d\}$. A nonempty subset of $\mathcal{I}$ is called an *itemset*. A transaction dataset D is a collection of itemsets, $D = \{t_1, \ldots, t_n\}$, where $t_i \subseteq \mathcal{I}$. For any itemset α, we denote the set of transactions that contain α as $D_\alpha = \{i | \alpha \subseteq t_i \ and \ t_i \in D\}$. Define the cardinality of an itemset α as the number of items it contains, *i.e.*, $|\alpha| = |\{o_i | o_i \in \alpha\}|$.

Definition 1 (Frequent Itemset) *For a transaction dataset D, an itemset α is* frequent *if $\frac{|D_\alpha|}{|D|} \geq \sigma$, where $\frac{|D_\alpha|}{|D|}$ is called the* support *of α in D, written $s(\alpha)$, and σ is the* minimum support threshold*, $0 \leq \sigma \leq 1$.*

We use *support set* to denote the set of transactions that contain a pattern, *i.e.*, D_α is the *support set* of α. A frequent itemset is called a *frequent pattern*, or a *pattern* for short in this paper. For two patterns α and α', if $\alpha \subset \alpha'$, then α is a *subpattern* of α', and α' is a *super-pattern* of α.

Definition 2 (Closed Frequent Pattern) *A frequent pattern α is* closed *if and only if it has no frequent super-pattern which has the same support set, i.e., for any frequent pattern α', if $\alpha \subset \alpha'$, then $D_\alpha \neq D_{\alpha'}$.*

Lemma 1 *For itemsets α and α', if $\alpha \subseteq \alpha'$, then $D_{\alpha'} \subseteq D_\alpha$.*

It is clear from Lemma 1 that for a pattern α, $D_\alpha = \bigcap_{\beta \subset \alpha} D_\beta$.

2.2 Robustness of colossal patterns

In this subsection, we show our observation on colossal patterns which is crucial for Pattern-Fusion. Our study on the relationship between the support set of a colossal pattern and those of its subpatterns reveals the notion of *robustness* of colossal patterns. Colossal patterns exhibit *robustness* in the sense that *if a small number of items are removed from the pattern, the resulting pattern would have a similar support set*. The larger the pattern size, the more prominent this robustness is observed. We capture this relationship between a pattern and its subpattern by the concept of *core pattern*.

Definition 3 (Core Pattern) *For a pattern α, an itemset $\beta \subseteq \alpha$ is said to be a τ-*core pattern *of α if $\frac{|D_\alpha|}{|D_\beta|} \geq \tau$, $0 < \tau \leq 1$. τ is called the core ratio.*

For a pattern α, let C_α be the set of all its core patterns, *i.e.*, $C_\alpha = \{\beta | \beta \subseteq \alpha, \frac{|D_\alpha|}{|D_\beta|} \geq \tau\}$ for a specified τ. In the rest of the paper, we would simply refer to a τ-core pattern as core pattern for brevity. With the definition of core pattern, we can formally define the robustness of a colossal pattern.

Definition 4 ((d,τ)-Robustness) *A pattern α is (d,τ)-*robust *if d is the maximum number of items that can be removed from α for the resulting pattern to remain a τ-core pattern of α, i.e.,*

$$d = \max_{\beta} \{|\alpha| - |\beta| | \beta \subseteq \alpha, \ and \ \beta \ is \ a \ \tau\text{-}core \ pattern \ of \ \alpha\}$$

Due to its robustness, a colossal pattern tend to have a large number of core patterns. Let α be a colossal pattern which is (d,τ)-robust. The following two lemmas show that the number of core patterns of α is at least exponential in d.

Lemma 2 *For a pattern $\beta \in C_\alpha$ and any itemset $\gamma \subseteq \alpha$, $\beta \cup \gamma \in C_\alpha$.*

Proof. It follows from Lemma 1 that $D_{\beta \cup \gamma} \subseteq D_\beta$, and as such, $\frac{|D_\alpha|}{|D_{\beta \cup \gamma}|} \geq \frac{|D_\alpha|}{|D_\beta|} \geq \tau$. By definition, we have $\beta \cup \gamma \in C_\alpha$. ■

Lemma 3 *For a (d,τ)-robust pattern α, $|C_\alpha| \geq 2^d$.*

Proof. For any β, such that $|\beta| = |\alpha| - d$, let U be the set of items in $\alpha \setminus \beta$, *i.e.*, U contains all the items that are in α but not in β. Then 2^U,the power set of U, is of size $2^{|\alpha|-|\beta|} = 2^d$. According to Lemma 2, for any itemset $t \in 2^U$, $\beta \cup t \in C_\alpha$. Hence, $|C_\alpha| > 2^d$. ∎

Since we observed that colossal patterns are more robust than patterns of smaller sizes, given a fixed core ratio τ, the set of core patterns of a colossal pattern is therefore much larger. Let's check an example. Figure 3 shows a

Transactions (# of Transactions)	Core Patterns ($\tau = 0.5$)
(abe) (100)	(abe),(ab),(be),(ae),(e)
(bcf) (100)	(bcf),(bc),(bf)
(acf) (100)	(acf),(ac),(af)
(abcef) (100)	(ab),(ac),(af),(ae),(bc),(bf),(be) (ce),(fe),(e),(abc),(abf),(abe) (ace),(acf),(afe),(bcf),(bce),(bfe) (cfe),(abcf),(abce),(bcfe),(acfe) (abfe),(abcef)

Figure 3. A transaction database and core patterns for each distinct transaction

simple transaction database with four different transactions each with 100 duplicates. $\{\alpha_1 = (abe), \alpha_2 = (bcf), \alpha_3 = (acf), \alpha_4 = (abcef)\}$. If we set $\tau = 0.5$, then, for example, (ab) is a core pattern of α_1 because (ab) is only contained by α_1 and α_4, thus $\frac{|D_{\alpha_1}|}{|D_{(ab)}|} = \frac{100}{200} \geq \tau$. α_1 is $(2, 0.5)$-robust while α_4 is $(4, 0.5)$-robust. The example shows that a larger pattern, *e.g.*, $(abcef)$, has far more core patterns than a smaller one, *e.g.*, (bcf).

The core pattern relationship can be extended to multiple levels by the definition of *core descendant*.

Definition 5 (Core Descendant) *For two patterns β and β', if there exists a sequence of $\beta_i, 0 \leq i \leq k, k \geq 1$ such that $\beta = \beta_0, \beta' = \beta_k$ and $\beta_i \in C_{\beta_{i+1}}$ for all $0 \leq i < k$, β is said to be a* core descendant *of β'.*

This core-pattern-based view of the pattern space leads to the following two observations which are essential in our algorithm design. In-depth exploration of these observations will be given in Section 3.

Observation 1. Due to the observation that a colossal pattern has far more core patterns than a smaller-sized pattern does, given a small c, a colossal pattern therefore has far more core descendants of size c. This means that a random draw from the complete set of patterns of size c would be more likely to pick a core descendant of a colossal pattern than that of a smaller-sized one. In Figure 3, consider the complete set of patterns of size $c = 2$ which contains $\binom{5}{2} = 10$ patterns in total, the probability of picking a core descendant of the colossal pattern $abcef$ on a random draw is 0.9, while the probability is at most 0.3 for all the other smaller-sized patterns.

Observation 2. A colossal pattern can be generated by merging a proper set of its core patterns. In fact, as any singe item o of the colossal pattern appears in more than one of its core patterns, o is missed only if all the core patterns containing o are absent in the set to be merged. For instance, $abcef$ can be generate by merging just two of its core patterns ab and cef, instead of merging all its 26 core patterns.

2.3 Pattern fusion overview

These observations on colossal patterns inspires the following mining approach: First generate a complete set of frequent patterns up to a small size, and then randomly pick a pattern, β. By our foregoing analysis β would with high probability be a core-descendant of some colossal pattern α. Identify all α's core-descendants in this complete set, and merge all of them. This would generate a much larger core-descendant of α, giving us the ability to leap along a path toward α in the core-pattern tree T_α. In the same fashion we pick K patterns. The set of larger core-descendants generated would be the candidate pool for the next iteration.

A question arises: Given β, which is a core-descendant of a colossal pattern α, how to find the other core-descendants of α? We first give the following pattern distance definition, with which we can show that two core patterns of a pattern α exhibit proximity in the corresponding metric space.

Definition 6 (Pattern Distance) *For patterns α and β, the pattern distance of α and β is defined to be $Dist(\alpha, \beta) = 1 - \frac{|D_\alpha \cap D_\beta|}{|D_\alpha \cup D_\beta|}$.*

Theorem 1 *[21] $(S, Dist)$ is a metric space, where S is a set of patterns and $Dist : S \times S \mapsto R^+$ is defined as in Definition 6.*

This means all the pattern distances satisfy the triangle inequality.

Theorem 2 *For two patterns $\beta_1, \beta_2 \in C_\alpha$, $Dist(\beta_1, \beta_2) \leq r(\tau)$, where $r(\tau) = 1 - \frac{1}{2/\tau - 1}$.*

Proof. Since both $\beta_1, \beta_2 \in C_\alpha$, we have

$$\begin{aligned} \frac{|D_{\beta_1} \cap D_{\beta_2}|}{|D_{\beta_1} \cup D_{\beta_2}|} &\geq \frac{|D_\alpha|}{|D_{\beta_1} \cup D_{\beta_2}|} \\ &= \frac{|D_\alpha|}{|D_{\beta_1}| + |D_{\beta_2}| - |D_{\beta_1} \cap D_{\beta_2}|} \\ &\geq \frac{|D_\alpha|}{|D_\alpha|/\tau + |D_\alpha|/\tau - |D_\alpha|} \\ &= \frac{1}{2/\tau - 1} \end{aligned}$$

Therefore, $Dist(\beta_1, \beta_2) = 1 - \frac{|D_{\beta_1} \cap D_{\beta_2}|}{|D_{\beta_1} \cup D_{\beta_2}|} \leq 1 - \frac{1}{2/\tau - 1} = r(\tau)$. ∎

It follows that all core patterns of a pattern α are bounded in the metric space by a "ball" of diameter $r(\tau)$. This means that given one core pattern $\beta \in C_\alpha$, we can identify all of α's core patterns in the current pool by posing a range query.

Note that the reverse direction of Theorem 2 is not true. In general, if $\beta_1 \in C_\alpha$ and $Dist(\beta_1, \beta_2) \leq r(\tau)$, it is not necessary the case that $\beta_2 \in C_\alpha$. In our mining algorithm, each randomly picked pattern could be a core-descendant of more than one colossal pattern, and as such, when merging the patterns found by the "ball", more than one larger core-descendant could be generated.

Now we are ready to give an overview of our mining model. Details of the algorithm will be presented in Section 4. Pattern-Fusion works in two phases.

1. **Initial Pool:** Pattern-Fusion assumes available an initial pool of small frequent patterns, which is the complete set of frequent patterns up to a small size, *e.g.*, 3. This initial pool can be mined with any existing efficient mining algorithm.
2. **Iterative Pattern Fusion:** Pattern-Fusion takes as input a user-specified parameter, K, which is the maximum number of patterns to be mined. The mining process is conducted iteratively. At each iteration, K seed patterns are randomly picked from the current pool. For each of these K seeds, we find all the patterns within a ball of a size specified by τ as defined in Definition 5. All the patterns in each "ball" are then fused together to generate a set of super-patterns. All the super-patterns thus generated are put together as a new pool. If this pool contains more than K patterns, the next iteration begins with this pool for the new round of random drawing. The termination of the iteration process is guaranteed by Lemma 1, as the support set of every super-pattern shrinks with each new iteration.

Note that *Pattern-Fusion merges all the small subpatterns of a large pattern in one step instead of expanding patterns with additional single items*. This gives Pattern-Fusion the advantage to circumvent mid-sized patterns and progress on a path leading to a potential colossal pattern. The idea is illustrated in Figure 4. Each point shown in the metric space represents a core pattern. A larger pattern has far more core patterns close to each other, all of which would be bounded by a ball as shown in dotted line, than a smaller pattern. Since the ball of the larger pattern is much denser, we will hit one of its core patterns with a higher probability when performing a random draw from the initial pattern pool.

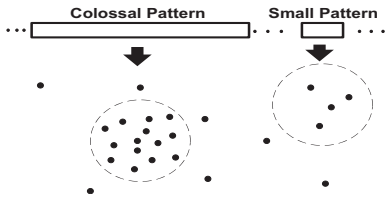

Figure 4. Pattern Metric Space

3 Towards Colossal Patterns

We show in this section why Pattern-Fusion could give a good approximation. First, we will show why Pattern-Fusion's mining result would favor colossal patterns over smaller-sized ones. Then we explore how Pattern-Fusion gives a good approximation by catching the outliers in the complete answer.

3.1 Why are colossal patterns favored?

In the last subsection, we have shown that a colossal pattern can be generated by merging just a subset of its core patterns. The message is that since any single item is likely to appear in a large number of core patterns, and so long as we grab one of these core patterns we won't miss the item, we will therefore be able to generate a colossal pattern with high probability. Let's look at a simple case to get a feel of the situation. Given a colossal pattern α of pattern size n and a drawing pool of size $\binom{n}{k}$ consisting of all k-tuples of the items in α, how large should a randomly-picked set from this pool be in order to recover α with high probability? The following theorem shows that it is actually a rather small set compared to the size of the drawing pool, which is $\binom{n}{k} \geq (n/k)^k$.

Theorem 3 *With probability at least $1 - 1/n^2$, a set of size $m^* = (en \ln n)/k$ picked uniformly at random will contain all items of α.*

Proof. Suppose the items of α are $o_1, o_2, \ldots, o_n$. Let $\xi_i(m^*)$ denote the event that item o_i is absent in a randomly picked set of size m^*, *i.e.*, none of the k-tuples contain o_i. Then the probability that α cannot be generated by such a set is $\Pr[\cup_{i=1}^{n} \xi_i(m^*)]$. Event $\xi_i(m^*)$ happens with a probability:

$$\begin{aligned}
\Pr[\xi_i(m^*)] &= \frac{\binom{\binom{n-1}{k}}{m^*}}{\binom{\binom{n}{k}}{m^*}} \\
&= \frac{\binom{n-1}{k}(\binom{n-1}{k}-1)\cdots(\binom{n-1}{k}-m^*+1)}{\binom{n}{k}(\binom{n}{k}-1)\cdots(\binom{n}{k}-m^*+1)} \\
&\leq \left(\frac{\binom{n-1}{k}}{\binom{n}{k}}\right)^{m^*} = \left(\frac{n-k}{n}\right)^{m^*} = \left(1-\frac{k}{n}\right)^{m^*}
\end{aligned}$$

Let $m^* = \lceil (3n \ln n)/k \rceil$, then $\Pr[\xi_i(m^*)] \leq 1/n^3$. By basic probability theory,

$$\Pr[\cup_{i=1}^{n} \xi_i(m^*)] \leq \sum_{i=1}^{n} \Pr[\xi_i(m^*)] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$$

Thus, we have established that with probability at least $1 - \frac{1}{n^2}$, no item of α will be missing in a randomly-picked set of size $m^* = (en \ln n)/k$, *i.e.*, α will be fully recovered. ■

In the last section, we have shown that a colossal pattern can be generated by merging just a subset of its core patterns. In particular, merging a set of *complementary* core patterns suffices.

Definition 7 (Complementary Core Pattern) *For a pattern α, a set $S \subseteq C_\alpha \setminus \{\alpha\}$ is a set of* complementary *core patterns of α if and only if $\bigcup_{\beta \in S} \beta = \alpha$.*

For example, in Figure 3, $\{(ab), (ae)\}$ is a set of complementary core patterns of (abe). For brevity, we simply call such a set S a complementary set when α is clear in the context. The set of all sets of complementary core patterns of α is denoted as Γ_α. If S — a set of complementary core patterns of α — appears in any iteration of the mining algorithm, and if any one pattern of S is picked by the random draw, then all the other core patterns in S will be found by the bounding "ball". Merging S would generate α.

Rationale. The more the number of such sets of complementary core patterns of α (*i.e.*, the larger the size of Γ_α), the greater the probability that α is generated. Figure 3 illustrates this point well.

Then the following lemma, immediate from Lemmas 2 and 3, shows that the number of complementary sets of a pattern is closely related to its robustness.

Lemma 4 *A (d, τ)-robust pattern α has at least $2^{d-1} - 1$ sets of complementary core patterns, i.e., $|\Gamma_\alpha| \geq 2^{d-1} - 1$.*

Rationale. Since our observation reveals that colossal patterns are more robust than those of smaller sized ones, this lemma means Pattern-Fusion would generate colossal patterns with greater probability. ■

There could be the case that there exist some small yet robust patterns. The following lemma reveals why, even for these small patterns, Pattern-Fusion also makes sure that most of them would not survive to appear in the final result.

Lemma 5 *Let l_i be the size of the smallest pattern in the pool at iteration i. Then $l_{i+1} \geq l_i$ for all i.*

Proof. By the construction algorithm of a new pattern pool, any pattern α in the pool at iteration $i + 1$ is the result of fusing a set of patterns in the pool at iteration i. Since the fusion operation takes the union of the patterns, the size of the new pattern α is at least as large as that of the smallest one in the fused set, which is in turn $\geq i$. ■

Due to Lemma 5, the patterns of the smallest size at each iteration will not be able to appear in the pool of the next iteration, unless they are picked by the random draw. For each of them, they survive to the next iteration with probability at most $\frac{K}{|S|}$ where S is the current pool. This means with high probability, after multiple iterations, small patterns will disappear from the current pool.

3.2 Catching the outliers

To evaluate the approximation quality, we introduce an evaluation model in Section 5 based on pattern edit distance. This distance definition gives a metric space on the patterns. Essentially, to give a mining result of size K which best approximate the complete set is to solve the K-Center problem in this metric space. Informally, given a metric space, the K-Center problem is to find the best K vertices to serve as centers such that the maximum over all distances from every vertex to its nearest center is minimized. As such, how well a subset of patterns approximate the complete answer set is measured by the maximum over all distance from every pattern in the complete set to its nearest neighbor in the subset. If this maximum is small, it means the subset well represents the complete answer in the sense that, for every pattern in the complete set, there exists in the subset some pattern which is close to it.

Evidently, to achieve a good approximation under this evaluation model, the mining algorithm would have to strive to catch those "outliers", *i.e.*, those patterns that are far from all the other patterns in the metric space, as missing one of them would entail significant approximation error. We show in the next theorem that one strength of Pattern-Fusion is the ability to catch "outliers"—the further away they lie, the more likely they will be generated.

Theorem 4 *Given the set U of all closed patterns for a transaction dataset D, a pattern $\alpha \in U$ and a core ratio τ, if the minimum pattern edit distance between α and any other pattern in U is d, then α is at least $(d - 1, \tau)$-robust.*

Proof. Since the minimum edit distance between α and any other pattern in U is d, then for all subpatterns $\beta \subseteq \alpha$ such that $|\alpha| - |\beta| < d$, we have $D_\beta = D_\alpha$. By Definition 4, α is hence at least $(d - 1, \tau)$-robust. ■

Combining Theorem 4 and Lemma 4, an outlier which is at edit distance d away from all others would have at least $2^{d-2} - 1$ sets of complementary core patterns. Hence, the further away it lies, the greater the chance that it will be generated by Pattern-Fusion.

4 Pattern-Fusion in Detail

Main Algorithm. The global algorithm is outlined in Algorithm 1. Lines 1 to 4 are the body of the iteration, which calls the algorithm Pattern_Fusion. After each iteration, it checks the frequent pattern set returned by Pattern_Fusion. If the result set contains more than K patterns, it begins the next iteration.

Algorithm 1 Main Algorithm

Input: Initial pool $InitPool$, Core ratio τ
Maximum number of patterns to mine K,
Output: Set of frequent patterns S
1: **do**
2: $S \leftarrow$ Pattern_Fusion($InitPool$, K,τ);
3: $InitPool \leftarrow S$
4: **while** $|S| > K$
5: **return** S;

Pattern Fusion. Pattern_Fusion randomly draws K seed patterns from the current pool. For each pattern α thus drawn, it examines the entire pool to find all the patterns that are within distance $r(\tau)$ from α, and records them in the set $\alpha.CoreList$. After all K patterns are drawn, a function Fusion($\alpha.CoreList$) fuses the patterns in each α's $CoreList$.

Line 1 initializes the result set S and the set T that will be used to record the K seed patterns. Lines 2 to 7 are the loop to pick the K seed patterns. Line 3 randomly draws a seed pattern from the current pool. Line 4 adds the drawn pattern α to T. Lines 5 to 7 examine every pattern in the current pool to find the "ball" for α. Lines 8 and 9 fuse the patterns in each "ball" and add the super-patterns generated by this fusion operation to the output set S.

The function $Fusion(\alpha.CoreList)$ fuses α and the patterns in $\alpha.CoreList$ to generate super-patterns. Since the reverse direction of Theorem 2 is not true, in general the patterns in $\{\alpha\} \cup \alpha.CoreList$ are core patterns of more than one pattern. $Fusion(\alpha.CoreList)$ generates the patterns β_i, such that for some subset $t_{\beta_i} \subseteq \alpha.CoreList$, $\{\alpha\} \cup t_{\beta_i} \in C_{\beta_{\beta_i}}$. When the number of such β_i exceeds a threshold, which is determined by the system, we resort to a sampling technique to decide the set of β_i to retain. The sampling is weighted on the size of t_{β_i}, which means β_i with a larger core pattern set would retain with higher probability. The set of β_i is generated by applying this sampling technique multiple times. This heuristic is based on the observation that patterns of larger size are likely to have an accordingly larger core pattern set. As such, this helps Pattern-Fusion to stay on paths which would be more likely to lead toward colossal patterns.

Algorithm 2 Pattern_Fusion

Input: Initial pool $InitPool$,Core ratio τ
Maximum number of patterns to mine K,
Output: Set of patterns S
1: $S \leftarrow \emptyset$; $T \leftarrow \emptyset$;
2: **for** $i = 1$ **to** K
3: Randomly draw a seed α from $InitPool$;
4: $T \leftarrow T \cup \{\alpha\}$
5: **for each** $\beta \in InitPool$
6: **if** $Dist(\alpha, \beta) \leq r(\tau)$
7: Record β in $\alpha.CoreList$
8: **for each** $\alpha \in T$
9: $S \leftarrow S \cup Fusion(\alpha.CoreList)$;
10: **return** S;

5 Evaluation Model

When the complete mining result is too huge to compute, a good approximation could be the only solution of exposing interesting patterns hidden in a large dataset. As the goal of our Pattern-Fusion method is to find patterns that have a global picture of the complete pattern set, traditional evaluation metrics in information retrieval like recall and precision no longer apply. In this sense, we propose an evaluation model that is able to capture how representative the mining result is.

Definition 8 (Itemset Edit Distance) *The edit distance $Edit(\alpha, \beta)$ between two itemsets α and β is defined as $Edit(\alpha, \beta) = |\alpha \cup \beta| - |\alpha \cap \beta|$.*

For example, the edit distance between itemsets $(abcd)$ and $(acde)$ is 2. Given two collections of itemsets, P and Q (P could be the mining result and Q be the complete pattern set), we need a measure to check how well P approximates Q. We developed a "clustering" model that is able to catch the semantics of patterns. Take each pattern $\alpha \in P$ as a cluster center and patterns $\beta \in Q$ as data points. Assign each data point in Q to clusters in P by performing a nearest-neighbor search. Definition 8 is taken as distance measure. For each cluster $i, 1 \leq i \leq |P|$ with center pattern $\alpha_i \in P$, let r_i be the edit distance between the farthest data point and the center pattern α_i. Then we regard $\frac{r_i}{|\alpha_i|}$ as the maximum approximation error for cluster i. The approximation error of P with respect to Q is the average of the maximum approximation errors of all the clusters. The formal definition is given as follows.

Definition 9 (Pattern Set Approximation) *For two pattern sets $P = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ and $Q = \{\beta_1, \beta_2, \ldots, \beta_n\}$, an* approximation *of P with respect to Q, denoted as A^P_Q, is a partition π_Q of Q, $\pi_Q = \{Q_1, Q_2, \ldots, Q_m\}$, such*

that $Q = \cup_{1 \le j \le m} Q_j, Q_i \cap Q_j = \emptyset$*, for* $i \neq j$*, and*

$$Q_i = \{\beta | \beta \in Q, \text{ and } Edit(\beta, \alpha_i) = \min_{1 \le j \le m} Edit(\beta, \alpha_j)\}$$

Definition 10 (Approximation Error) *The* approximation error *of an approximation* A_Q^P *is denoted as* $\Delta(A_Q^P)$*,*

$$\Delta(A_Q^P) = \frac{\sum_{i=1}^{m} r_i}{m}$$

where $r_i = \max_{\beta \in Q_i} \frac{Edit(\beta, \alpha_i)}{|\alpha_i|}$

The approximation error $\Delta(A_Q^P)$ gives the average maximum pattern distance between *any* pattern in the complete set Q and some pattern in the mining result set P. Hence, the smaller the approximation error, the better P approximates Q, in the sense that P has better representatives of the patterns in Q. See the following example.

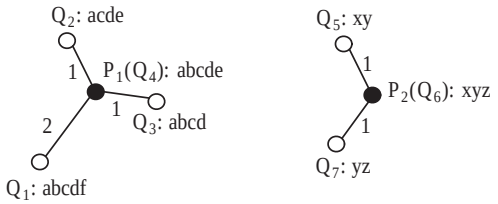

Figure 5. Pattern Set Approximation A_Q^P

Example 1 *Suppose there is a complete set* $Q = \{Q_1, \ldots, Q_7\}$*, as shown in Figure 5. The approximation set* $P = \{P_1, P_2\}$*, where* $P_1 = Q_4$*, and* $P_2 = Q_6$*. By definition,* Q_1 *is the pattern with the largest distance from* P_1 *in* P_1*'s cluster. Since* $Edit(Q_1, P_1) = 2$ *and* $|P_1| = 5$*, the approximation error of* P_1 *equals* $\frac{2}{5}$*. Similarly,* Q_5 *and* Q_7 *are of equal distance to* P_2 *in* P_2*'s cluster. Since* $Edit(Q_5, P_2) = 1$ *and* $|P_2|$*=3,* $r_2 = \frac{1}{3}$*. Therefore,* $\Delta(A_Q^P) = (\frac{2}{5} + \frac{1}{3})/2 = \frac{11}{30} = 0.37$*. This means, on average, any pattern in* Q *is at most* $0.37 \times 5 \approx 2$ *items away from some pattern in* P*.*

6 Experimental Results

In this section, we are going to demonstrate the efficiency and effectiveness of the Patter-Fusion method. All of the experiments are performed on a 3.2GHZ, 1GB-memory, Intel PC running Windows XP.

in our experiments, we included one synthetic dataset and two real datasets. The real datasets are built from program tracing data and microarray data.

Synthetic data set. To illustrate the situation when the mining result contains an exponential number of frequent patterns. We use the example $Diag_n$ given in the introduction section. $Diag_n$ is a $n \times (n-1)$ table where the i_{th} row has integers from 1 to n except i. Each row is taken as an itemset. The minimum support threshold is set at $n/2$. Figure 6 shows the running time of Pattern-Fusion against LCM_maximal [18], which is a maximal pattern mining algorithm. It is easy to observe that as n increases, the running time of LCM_maximal increases exponentially since the number of patterns equals $\binom{n}{\frac{n}{2}}$, rendering the time cost unaffordable even for a moderate value of n. Therefore, instead of reporting the complete set, an approximation mining result is more appropriate for this scenario. Figure 7 shows the approximation errors of Pattern-Fusion running on $Diag_{40}$ with minimum support 20. Pattern-Fusion starts with an initial pool of 820 patterns of size ≤ 2. The mining result is compared with the complete set S each of which is a pattern of size 20. Since the complete set S is too big, the complete set is randomly sampled for comparison. It is observed that Pattern-Fusion has comparable approximation error as a uniform sampling approach, which randomly picks up K patterns from the complete answer set. It means Pattern-Fusion will not get stuck locally during the mining.

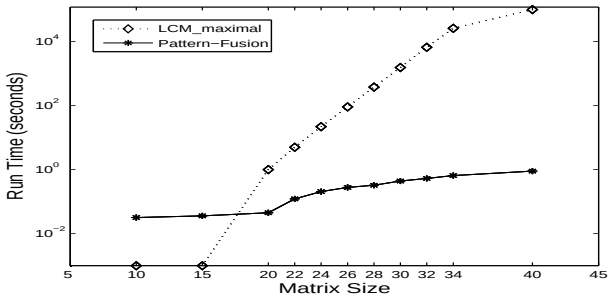

Figure 6. Run Time on $Diag_n$

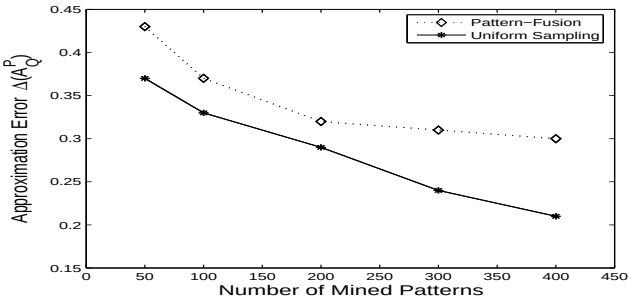

Figure 7. Approximation Error on $Diag_n$

Real data set 1: Replace. Replace is a program trace data set collected from the "replace" program, which is one of the *Siemens Programs* that have been widely used in software engineering research [12]. The program calls and tran-

sitions of 4,395 correct executions are recorded. Each type of program calls and transitions is considered as one item. There are 66 different program calls and transitions in total. The purpose of finding frequent patterns in this data set is to identify frequent, and accordingly normal, program execution structures, which will be compared against abnormal program execution structure in an attempt to isolate program bugs.

The Replace data set contains 4,395 transactions. There are 57 items in total. With a minimum support threshold of 0.03, the complete set of frequent patterns in Replace contains 4,315 patterns. There are three largest patterns with size 44. We notice that in all the experiments conducted on Replace, with different settings of K and τ, Pattern-Fusion is always able to find all these three colossal patterns. We start with an initial pool of 20,948 patterns of size ≤ 3.

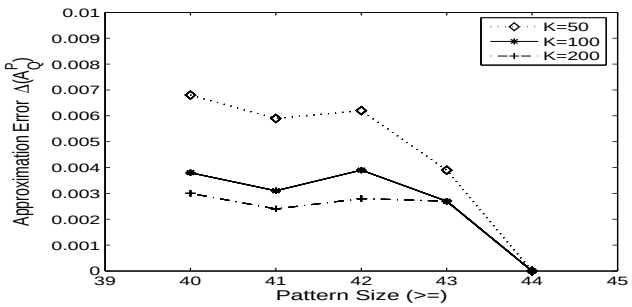

Figure 8. Approximation Error on Replace

Figure 8 illustrates the approximation error $\Delta(A_Q^P)$ of Pattern-Fusion's mining result (P) compared against the complete set (Q), showing that Pattern-Fusion's mining result is a good approximation of the complete set. A data point with coordinates $(\hat{x}, \hat{y})$ means the approximation error $\Delta(A_Q^P)$ is $\hat{y}$, when our mining result is compared with the complete set for all patterns of size $\geq \hat{x}$. For example, there are totally 98 closed frequent patterns of size ≥ 42 in the complete set. When $K = 100$, Pattern-Fusion returns 80 of them. The corresponding data point is $(42, 0.0039)$, which means these 80 patterns represent the complete set well such that any pattern in the complete set is on average at most $44 \cdot 0.0039 = 0.17$ items in difference from one of these 80 patterns. The largest ones, those of size 44, are never missed. It is clear from Figure 8 that better approximations are achieved if more seed patterns are selected.

Real data set 2: ALL. ALL is a popular gene expression data set. It is a clinical data on ALL-AML leukemia[1]. Each item is a column, which represents the activity level of genes/proteins in the sample. Frequent patterns in the data would reveal important correlations between gene expression patterns and disease outcomes, offering researchers clinically useful diagnostic knowledge.

The ALL data set contains 38 transactions, each of size 866. There are 1736 items in total. We first show that when minimum support threshold is high (*e.g.*, 30), Pattern-Fusion generates mining results of high quality. We start with an initial pool of 25,760 patterns of size ≤ 2. Figure 9 compares Pattern-Fusion's mining result ($K = 100$) against the complete set for frequent patterns of size > 70, which are the colossal ones for this data. In fact, Pattern-Fusion is able to get all the largest colossal patterns with size greater than 85.

Pattern Size	110	107	102	91	86	84	83
The complete set	1	1	1	1	1	2	6
Pattern-Fusion	1	1	1	1	1	1	4
Pattern Size	82	77	76	75	74	73	71
The complete set	1	2	1	1	1	2	1
Pattern-Fusion	0	2	0	1	1	1	1

Figure 9. Mining Result Comparison on ALL

Figure 10 shows the running time for three mining algorithms with decreasing minimum support threshold. Both LCM_maximal [18] and TFP (top-k) [19] suffer from exponentially increasing running time as the minimum support threshold decreases, while the running time of Pattern-Fusion levels off.

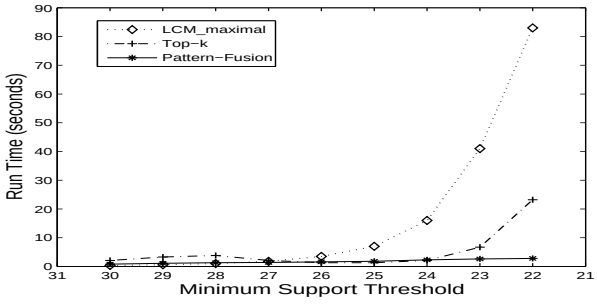

Figure 10. Run Time on ALL

7 Related Work

Frequent itemset mining, initiated by the introduction of association rule mining [1], has been extensively studied [2, 17, 10, 4, 3, 11, 22, 13]. Efficient implementations appeared in the FIMI workshops. Most of the well studied frequent pattern mining algorithms, including Apriori [2], FP-growth [11], and CARPENTER [15], mine the complete set of frequent itemsets.

[1] http://www.broad.mit.edu/tools/data.html

According to the Apriori property, any subset of a frequent itemset is frequent. This downward closure property leads to an explosive number of frequent patterns. The introduction of closed frequent itemsets [16] and maximal frequent itemsets [9, 3] can partially alleviate this redundancy problem. Extensive studies have proposed fast algorithms for mining frequent closed itemsets, such as A-close [16], CHARM [22] and CLOSET+ [20], and maximum closed itemsets, such as, Max-Miner [3], MAFIA [5] and GenMax[7].

In all of these studies, the mining of the complete pattern set becomes the major task. While in many applications, there exist an explosive number of closed or maximum patterns, none of the existing algorithms is able to complete the mining in a reasonable amount of time. To the best of our knowledge, ours is the first work to acknowledge the existence of this problem and provide a novel solution. Furthermore, our proposed quality measure system provides a benchmark to evaluate any partial mining result.

8 Conclusions

We studied the problem of efficient computation of a good approximation for the colossal frequent itemsets in the presence of an explosive number of frequent patterns. Based on the concept of core pattern, a new mining methodology, Pattern-Fusion, is introduced. An evaluation model is proposed to evaluate the approximation quality of the mining results of Pattern-Fusion against the complete answer set. This model also provides a general mechanism to compare the difference between two sets of frequent patterns. Empirical studies conducted on both synthetic and real data sets demonstrated that Pattern-Fusion is able to give good approximation for colossal patterns in data sets that no existing mining algorithm can. This paper is an initial effort toward mining colossal frequent patterns in more complicated data, such as sequences and graphs, where the essential idea developed in this paper could be applied.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In SIGMOD'93, pages 207–216.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In VLDB'94, pages 487–499.

[3] R. Bayardo. Efficiently mining long patterns from databases. In SIGMOD'98, pages 85–93.

[4] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket analysis. In SIGMOD'97, pages 255–264.

[5] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In ICDE'01, pages 443–452.

[6] G. Cong, K. Tan, A. K. H. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In SIGMOD'05, pages 670–681.

[7] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In ICDM'01, pages 163–170.

[8] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In FIMI'03.

[9] D. Gunopulos, H. Mannila, R. Khardon, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In PODS'97, pages 209 – 219.

[10] D. Gunopulos, H. Mannila, and S. Saluja. Discovering all most specific sentences by randomized algorithms. In ICDT'97, pages 215–229.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In SIGMOD'00, pages 1–12.

[12] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In ICSE'94, pages 191–200.

[13] J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent item sets by opportunistic projection. In KDD'02, pages 239–248.

[14] H. Mannila, H Toivonen, and A. Verkamo. Efficient algorithms for discovering association rules. KDD'94, pages 181–192.

[15] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. Zaki. CARPENTER: Finding closed patterns in long biological datasets. In KDD'03, pages 637–642.

[16] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In ICDT'99, pages 398–416.

[17] H. Toivonen. Sampling large databases for association rules. In VLDB'96, pages 134–145.

[18] T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In FIMI'04.

[19] J. Wang, J. Han, Y. Lu, and P. Tzvetkov. TFP: An efficient algorithm for mining top-k frequent closed itemsets. TKDE, 17:652–664, 2005.

[20] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In KDD'03, pages 236–245.

[21] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In VLDB'05, pages 709–720.

[22] M. Zaki and C. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In SDM'02, pages 457–473.