Mining Closed Discriminative Dyadic Sequential Patterns

David Lo[♯], Hong Cheng[§], and Lucia[♯]
[♯]School of Information Systems, Singapore Management University
[§]Department of Systems Engineering and Engineering Management,
The Chinese University of Hong Kong
{davidlo,lucia.2009}@smu.edu.sg,hcheng@se.cuhk.edu.hk

ABSTRACT

A lot of data are in sequential formats. In this study, we are interested in sequential data that goes in pairs. There are many interesting datasets in this format coming from various domains including parallel textual corpora, duplicate bug reports, and other pairs of related sequences of events. Our goal is to mine a set of closed discriminative dyadic sequential patterns from a database of sequence pairs each belonging to one of the two classes +ve and -ve. These dyadic sequential patterns characterize the discriminating facets contrasting the two classes. They are potentially good features to be used for the classification of dyadic sequential data. They can be used to characterize and flag correct and incorrect translations from parallel textual corpora, automate the manual and time consuming duplicate bug report detection process, etc. We provide a solution of this new problem by proposing new search space traversal strategy, projected database structure, pruning properties, and novel mining algorithms. To demonstrate the scalability and utility of our solution, we have experimented with both synthetic and real datasets. Experiment results show that our solution is scalable. Mined patterns are also able to improve the accuracy of one possible downstream application, namely the detection of duplicate bug reports using pattern-based classification.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Data Mining

1. MOTIVATION & INTRODUCTION

A lot of data are in sequential formats. Some examples include series of words in a textual document, DNA sequences, protein sequences, purchase histories, program execution traces, etc. In these data sources each data unit is a sequence of atomic events. For example, in textual corpora, each document is a sequence of words. Much of this data could be analyzed to extract important knowledge useful to decision makers.

In this work, we are interested in the analysis of sequential data. We specifically consider data in the form of pairs of sequences. Each data unit is composed of two sequences, and we consider a database of such sequence pairs. Our data representation is motivated by many data sources in this particular data format from various domains including software engineering, text mining, social network, etc. Thus, an algorithm that could analyze such data sources potentially leads to many real-world applications. We illustrate some examples in the following paragraphs.

In the software engineering domain, there are important problems requiring the analysis of pairs of sequences. One of such problems is the detection of duplicate bug reports [22]. Duplicate bug reports are parallel textual documents expressed in different ways. Daily, large software vendors, e.g., Microsoft, receive millions of bug reports from their clients. Due to the flexibility of natural language, many bug reports, referring to the same problem, could be expressed in various ways. This causes a problem as software engineers (also referred to as bug triagers) are flooded with a large mass of bug reports to be checked. Mining patterns that discriminate pairs of bug reports that are duplicate from those that are not would be very useful to address this problem.

In the text mining domain, recently there is an active interest on automated machine translations [19]. A translation is simply a mapping of two documents, each being a sequence of word tokens describing the same content in different ways. Some translations are bad while others are good. It would be interesting to find some rules that differentiate good from bad translations. There are many books describing common language error patterns, e.g., [14]. Mining common good or bad translation patterns would help in automated machine translation work, where the translation process is performed by a machine.

One could imagine many other data expressed as a pair of sequences, e.g., a pair of program traces representing the activities or behaviors of two related sub-systems, a pair of sequences of activities performed by two accomplices in a fraud case (e.g., Enron), etc.

All the above problems necessitate a need to mine from pairs of sequences. Class labels could be attached to these pairs, e.g., duplicate or not, good or bad translation, correct or erroneous program execution, fraud or not, etc. All the above problems would benefit from a mining engine that could extract features that discriminate good from bad pairs. These features are in the form of pairs of sequential patterns referred to in this paper as *dyadic sequential patterns*. In particular, we are interested to mine patterns that are both *frequent* and *discriminative*. Also, rather than mining

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EDBT 2011, March 22–24, 2011, Uppsala, Sweden.
Copyright 2011 ACM 978-1-4503-0528-0/11/0003 ...$10.00

all frequent and discriminative patterns, it would be better to mine a compact set of patterns, for example, closed patterns [27, 24] which are a lossless compression of all frequent patterns. We refer to our problem as mining *closed discriminative dyadic sequential patterns*.

Mining dyadic sequential patterns poses a number of technical challenges. First, there is a need to define the search space and specify a good search space traversal strategy so that no pattern is visited more than once or missed. Second, there is a need to design a good data structure for efficient mining of dyadic sequential patterns. Different from a standard sequential pattern, a dyadic pattern consists of two sequences and there is no restriction that the left sequence of the pattern needs to match the left sequence of a sequence pair in the database. Indeed, it could match either the left or the right sequence. There is also a need to identify new properties that could be used to prune the search space of infrequent, non-closed, and non-discriminative dyadic patterns. It would be best if a *direct* mining strategy could be performed where non-interesting patterns are removed early rather than late (i.e., via a post-mining filtering step).

We address the above challenges in our proposed mining solution. At a high-level view, which would be made concrete in subsequent sections, our algorithm traverses the search space of possible patterns in a depth first search fashion by extending smaller patterns to larger ones. We employ a traversal strategy to ensure that no patterns are visited more than once. Also, all interesting (i.e., frequent, closed, and discriminative) patterns would be visited. During the traversal, pattern statistics are computed efficiently and incrementally by leveraging a novel data structure. Search spaces containing non-interesting patterns are pruned to significantly reduce the number of unfruitful checks of bad pattern candidates. In the process, frequent, closed, and discriminative patterns are output when they are found.

We experiment our solution on a number of synthetic datasets to test the sensitivity of our approach on various dataset parameters. We also experiment our solution on a real bug report dataset and show that we could mine interesting patterns which could help to improve the accuracy in detecting duplicate bug reports.

The contributions of this work are as follows:

1. We propose a new problem of mining closed discriminative dyadic sequential patterns and show the potential applications.
2. We propose a new algorithm that utilizes a new search space traversal strategy, a new data structure, and several new and adapted pruning strategies.
3. We show the scalability of our solution on several synthetic and real datasets.
4. We show the utility of our solution in improving a downstream application task, in particular the detection of duplicate bug reports.

This paper is structured as follows. Section 2 describes related work. Section 3 formalizes some important concepts and definitions. Section 4 describes our new search space traversal strategy. Our data structure is presented in Section 5. We state several properties and theorems in Section 6. We elaborate several variants of our proposed algorithm in Section 7. Experimental results on synthetic and real datasets are presented in Section 8. We conclude and discuss future work in Section 9.

2. RELATED WORK

Agrawal and Srikant proposed frequent itemset mining in [4]. Given a transaction database, the task is to find itemsets that are frequent in the database based on a user defined minimum support threshold. There have been a number of studies extending frequent itemset mining to mine a set of closed itemsets [25], improve mining speed [23], etc.

Agrawal and Srikant later extended frequent itemset mining to sequential pattern mining in [5]. In sequential pattern mining, frequent patterns are mined from sequences. Hence, there is a temporal order among elements in a pattern. Given a sequence database, the task is to find sequential patterns that appear in no less than a minimum number of sequences. There have been a number of studies extending standard sequential pattern mining, including mining closed patterns [27, 24], generators [12, 16], repetitive patterns [17, 10], non-redundant rules [18], discriminative sequential patterns [15], as well as parallel sequential pattern mining [9]. In this work, we extend sequential pattern mining by mining dyadic sequential patterns. A dyadic sequential pattern comprises of two sequences. The new problem setting requires new search space traversal strategy, data structure, pruning strategies, and algorithms.

Dong and Li propose emerging patterns in [11], which are patterns that appear more frequently in one class than the other. Cheng et al. propose discriminative itemset patterns based on information gain and show that these patterns are useful for improving classification accuracy [7]. Cheng et al. later extend their approach to directly mine discriminative patterns with a branch-and-bound search strategy [8]. Lo et al. mine discriminative sequences from program execution traces [15]. Yan et al. mine discriminative graphs from a graph database [26]. Similar to the work by Lo et al. [15], we mine discriminative sequential patterns. But the patterns we are interested in are *dyadic* sequential patterns where each pattern consists of two sequences. Besides, in [15] non-discriminative patterns are removed during a post-mining filtering step. In this work, we push a pruning strategy based on a few interesting properties of the discriminative score deep into the mining process, which prunes the search space and significantly improves the mining efficiency and scalability.

3. CONCEPTS & DEFINITIONS

In this section, we describe some concepts and definitions on sequence database, dyadic sequential pattern, and discriminative score.

DEFINITION 3.1 (**Sequence**). *A sequence is a series of events from an alphabet* Δ. *We assume that there is a linear order among the events in the alphabet. This linear order could be alphabetic order, numeric order, etc. We denote a sequence of length* n *by* $\langle e_1, e_2, \ldots, e_n \rangle$.

DEFINITION 3.2 (**Subsequence Relation**). *A sequence* $s1 = \langle e_1, ..., e_n \rangle$ *is a subsequence of another sequence* $s2 = \langle f_1, \ldots, f_m \rangle$, *iff there exist integers* $1 \leq x_1 \leq \ldots \leq x_n \leq m$ *such that* $\forall_{i=1 \ldots n}\ e_i = f_{x_i}$. *We denote this as* $s1 \sqsubseteq s2$.

DEFINITION 3.3 (**Sequence Pair with Label**). *A sequence pair consists of two sequences. We denote this as s1-s2, where s1 and s2 are sequences. The left sequence of a sequence pair s is denoted as s.Left, and the right one is denoted as s.Right. In our setting, we attach a class label to a*

sequence pair which is either +ve or -ve. These abstract labels correspond to two contrasting user-defined domain-specific concepts, e.g., fraud vs no fraud, duplicate vs non-duplicate, etc.

DEFINITION 3.4 (**Subsequence Relation Over Pair**). *Consider two pairs s = s1-s2 and r = r1-r2, we say that s is a subsequence of r iff s1 $\sqsubseteq$ r1 and s2 $\sqsubseteq$ r2. We denote this as s $\sqsubseteq_p$ r.*

DEFINITION 3.5 (**Sequence Pair Database**). *A sequence pair database is a multi-set of sequence pairs denoted as DB. The size of a database is the number of sequence pairs it contains and is denoted as |DB|. The number of pairs belonging to the +ve class and -ve class in a database DB is denoted as DB.Pos and DB.Neg respectively.*

DEFINITION 3.6 (**Dyadic Sequential Pattern**). *A dyadic sequential pattern P is a pair of sequences. It is denoted as p1-p2, where the first sequence p1 is referred to as the left sequence of P, while p2 is referred to as the right sequence of P. The left sequence of the pattern P is denoted as P.Left, and the right one is denoted as P.Right.*

DEFINITION 3.7 (**Pattern Occurrence**). *Consider a dyadic sequential pattern P=p1-p2 and a sequence pair S=s1-s2. P occurs in S, iff p1 is a sub-sequence of s1 and p2 is a subsequence of s2. In other words, P $\sqsubseteq_p$ S.*

DEFINITION 3.8 (**Pattern Frequency/Support**). *Given a sequence pair database DB, the frequency or support of a pattern P in database DB is the number of sequence pairs in DB where P occurs. We denote the support of a pattern P in DB by sup(P,DB). The number of +ve labeled sequences in DB where P occurs is denoted as sup_{+ve}(P,DB). The number of -ve labeled sequences in DB where P occurs is denoted as sup_{-ve}(P,DB). It must be the case that sup(P,DB) = sup_{-ve}(P,DB) + sup_{+ve}(P,DB). We drop DB if it is clear from the context.*

DEFINITION 3.9 (**Discriminative Score**). *Consider a pattern r in a database DB. The discriminative score of r is defined by the information gain [21] as:*

$$IG(c|r) = H(c) - H(c|r) \quad (1)$$

where $H(c) = -\sum_{c_i \in \{\pm ve\}} P(c_i) \log P(c_i)$ *is the entropy and* $H(c|r) = -\sum P(r) \sum_{c_i \in \{\pm ve\}} P(c_i|r) \log P(c_i|r)$ *is the conditional entropy given the pattern r. The discriminative score of a pattern r in DB is denoted as disc(r,DB). We drop DB if it is clear from the context.*

If we use the notations $DB.Pos$, $DB.Neg$, $sup_{+ve}(r, DB)$ and $sup_{-ve}(r, DB)$ in the information gain calculation, we have the following expressions. Let $p = sup_{+ve}(r, DB)$, $q = sup_{-ve}(r, DB)$ and $|DB| = DB.Pos + DB.Neg$, then

$$H(c) = -\frac{DB.Pos}{|DB|} \log \frac{DB.Pos}{|DB|} - \frac{DB.Neg}{|DB|} \log \frac{DB.Neg}{|DB|}$$

$$\begin{aligned} H(c|r) = & - \frac{p+q}{|DB|}\left(\frac{p}{p+q} \log \frac{p}{p+q} + \frac{q}{p+q} \log \frac{q}{p+q}\right) \\ & - \frac{|DB|-(p+q)}{|DB|}\left(\frac{DB.Pos-p}{|DB|-(p+q)} \log \frac{DB.Pos-p}{|DB|-(p+q)}\right. \\ & + \left.\frac{DB.Neg-q}{|DB|-(p+q)} \log \frac{DB.Neg-q}{|DB|-(p+q)}\right) \end{aligned}$$

Idx	Sequence Pair	Label
1	$\langle a,b,d,d\rangle - \langle e,c,d,d,e\rangle$	+ve
2	$\langle a,b,d,d\rangle - \langle e,c,d,d,e\rangle$	+ve
3	$\langle a,b,d,d\rangle - \langle e,c,d,d,e\rangle$	+ve
4	$\langle a,a,b,d,d\rangle - \langle e,c,d,d,e\rangle$	+ve
5	$\langle b,c,d,d\rangle - \langle e,f,g\rangle$	+ve
6	$\langle a,b,d,d\rangle - \langle e,c,d,d,e\rangle$	-ve
7	$\langle a,b,d,d\rangle - \langle e,d,c,d,e\rangle$	-ve
8	$\langle a,b,d,d\rangle - \langle c,d,d\rangle$	-ve
9	$\langle a,d,d\rangle - \langle e,c,d,e,d\rangle$	-ve

Table 1: Example Database ExDB

Num	Pattern P	sup(P)	disc(P)
1	$\langle a\rangle$ - $\langle d\rangle$	8	0.102
2	$\langle a,d\rangle$- $\langle d\rangle$	8	0.102
3	$\langle a,d,d\rangle$ - $\langle d\rangle$	8	0.102
	...		
4	$\langle a\rangle$ - $\langle e,c,d,d,e\rangle$	5	0.229
5	$\langle a,b\rangle$ - $\langle e,c,d,d,e\rangle$	5	0.229
6	$\langle a,b,d\rangle$ - $\langle e,c,d,d,e\rangle$	5	0.229
	...		
7	$\langle b\rangle$ - $\langle e\rangle$	7	0.320
8	$\langle b,d\rangle$ - $\langle e\rangle$	7	0.320
9	$\langle b,d,d\rangle$ - $\langle e\rangle$	7	0.320
	...		

Table 2: Some Mined Frequent Discriminative Patterns from ExDB @ min_sup = 4, min_disc = 0.1

Given a sequence pair database DB, $DB.Pos$ and $DB.Neg$ are fixed values. Thus the information gain of a pattern P is a function of $sup_{-ve}(P, DB)$ and $sup_{+ve}(P, DB)$, i.e.,

$$disc(P, DB) = IG(sup_{+ve}(P, DB), sup_{-ve}(P, DB))$$

DEFINITION 3.10 (**Frequent Pattern**). *A pattern is frequent if its support is no less than a user defined threshold min_sup.*

DEFINITION 3.11 (**Discriminative Pattern**). *A pattern is discriminative if its discriminative score is no less than a user defined threshold min_disc.*

DEFINITION 3.12 (**Closed Pattern**). *A pattern p1 is closed if there does not exist another pattern p2 with the same support and discriminative score, where either one of the following conditions holds:*

$$(Cond\ 1)\ p1.Left \sqsubseteq p2.Left \wedge p1.Right \sqsubseteq p2.Right$$

$$(Cond\ 2)\ p1.Left \sqsubseteq p2.Right \wedge p1.Right \sqsubseteq p2.Left$$

In this paper, when we refer to closed patterns, unless otherwise stated, the patterns are also frequent.

Problem Definition. Given a sequence pair database DB, a minimum support threshold min_sup, and a minimum discriminativeness threshold min_disc, find all closed, frequent, and discriminative dyadic sequential patterns.

Example. Consider the database shown in Table 1. Some frequent and discriminative patterns mined with $min_sup = 4$ and $min_disc = 0.1$ are shown in Table 2. The set of closed patterns is shown in Table 3. There are totally 52 frequent and discriminative patterns; out of those, only 5 are closed. Many patterns are non-closed due to another longer pattern with the same support and discriminative score. For example, the patterns numbered 1, 2, and 3 in Table 2 are subsumed by the first pattern in Table 3.

Num	Pattern P	sup(P)	disc(P)
1	$\langle a, d, d\rangle$ - $\langle d, d\rangle$	8	0.102
2	$\langle a, b, d, d\rangle$ - $\langle e, c, d, d, e\rangle$	5	0.229
3	$\langle a, d, d\rangle$ - $\langle c, d\rangle$	8	0.102
4	$\langle b, d, d\rangle$ - $\langle e\rangle$	7	0.320
5	$\langle d, d\rangle$ - $\langle e\rangle$	7	0.143

Table 3: Mined Closed Discriminative Patterns from ExDB @ min_sup = 4, min_disc = 0.1

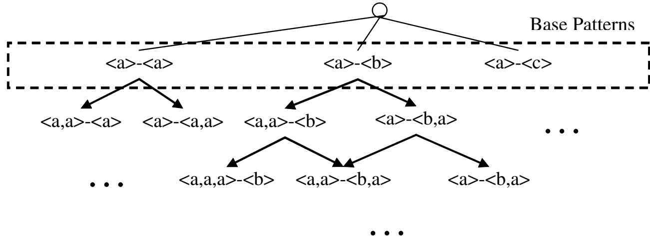

Figure 1: Basic Search Space Traversal

4. SEARCH SPACE TRAVERSAL

To mine for closed, frequent, and discriminative patterns, we need to characterize and traverse the search space of all possible patterns. In this section, we describe several search space traversal strategies from the most basic to the one that we eventually use in this paper. In the process, we highlight the challenges of designing a good search space traversal strategy for mining dyadic sequential patterns.

4.1 Basic Traversal

The search space of all possible patterns are all possible pairs of sequences. A basic approach to traverse all the patterns in this search space is to start with patterns of size two, i.e., a pair of sequences each containing one event. Each of these patterns could then be grown by appending events to the left and right sequences of the pattern. We call the resultant patterns grown from a pattern P as P's *descendants*. This is illustrated in Figure 1 where each directed arrow or edge in the search space lattice corresponds to a pattern growth operation. All the patterns could be generated by traversing all edges in the search space.

4.2 Preventing the Generation of Redundant Patterns

There is an issue with the basic approach as some patterns are visited more than once. For example, all patterns rooted at node $\langle a, a\rangle - \langle b, a\rangle$ in Figure 1 would be visited multiple times. To address this issue we need to eliminate some edges in the search space lattice. Let us label the edges as L and R, to represent the expansion of the left sequence or the right sequence of a pattern respectively. We refer to the resultant patterns generated by traversing the L and R edges as the *left-extension patterns* and *right-extension patterns* respectively.

To ensure that all nodes in the search space is visited only once, for every node visited though the traversal of L edges, we enforce that only L edges could be traversed in the subsequent pattern growth operations. By this rule, every node in the search space would only be visited once. We would not miss any patterns, as every pattern could be grown from a size-2 pattern by first performing right-extensions followed by left-extensions. The traversal scheme is shown in Figure 2.

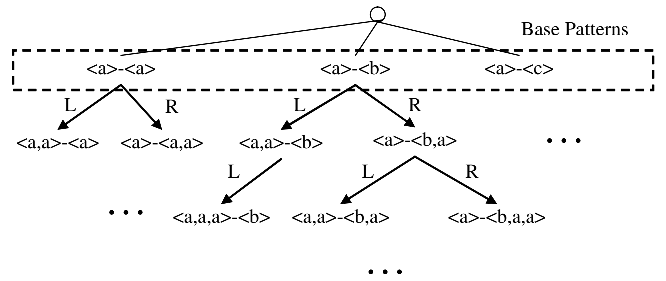

Figure 2: Non-Redundant Search Space Traversal

4.3 Preventing the Generation of Isomorphic Patterns

Another issue arises due to pattern isomorphisms. Note that the dyadic pattern $\langle a, b\rangle - \langle c, d\rangle$ is isomorphic to the pattern $\langle c, d\rangle - \langle a, b\rangle$ and they both have the same meaning. The second pattern is just the first pattern where the left sequence is made the right and the right is made the left.

To prevent the generation of isomorphic patterns, we first define a lexicographic ordering on sequences in Definition 4.1. We then enforce a canonical representation of pattern as specified in Definition 4.2.

DEFINITION 4.1 (**Sequence Lexicographic Order**). *Each event comes from an alphabet with elements having a linear order. Consider two sequences $A = \langle a_1, a_2, \ldots, a_n\rangle$ and $B = \langle b_1, b_2, \ldots, b_m\rangle$. We say that $A < B$, if and only if one the following conditions holds:*

$$(Cond\ 1)\ \exists_{0<y\leq\ min(m,n)}.(\forall_{x<y}.a_x = b_x) \wedge (a_y < b_y)$$

$$(Cond\ 2)\ \forall_{x\leq n}.(a_x = b_x) \wedge (n < m)$$

DEFINITION 4.2 (**Canonical Pattern**). *A pattern $P = \langle A, B\rangle$ is a canonical pattern iff the left sequence A is lexicographically smaller than or equal to the right sequence B, i.e., $A \leq B$.*

Next, we state a useful property of canonical patterns in Property 1. The implication of Property 1 is as follows: All patterns grown from a non-canonical pattern would not be canonical. Thus, we could avoid generating and checking many patterns based on this property.

PROPERTY 1 (**Canonical Pruning**). *A canonical left-extension pattern can only be grown from a canonical left- or right- extension pattern. A canonical right-extension pattern can only be grown from a canonical right-extension pattern.*

PROOF. *Part 1: Left-Extension. Consider for contradiction that a canonical left-extension pattern Q is grown from a non-canonical left- or right-extension pattern P. There are two conditions where Q is deemed canonical.*

The first condition requires the existence of an element x in the left sequence of Q to be smaller than the corresponding element having the same index in the right sequence, where all other previous elements are the same. As Q is grown from P, P must be a smaller pattern. As Q is a left extension pattern, they must have the same right sequence. Thus there are again two cases. The first case is such that x exists in the left sequence of P; if this is the case, then P is canonical too and thus a contradiction. The second case is such that x does not exist in the left sequence of P implying that all elements of the left sequence are the same as corresponding ones (i.e., those having the same indices) in the right sequence, also it

must be the case that the length of the right sequence is more than that of the left. Thus, there is also a contradiction.

The second condition requires that all elements of the left sequence of Q match with the corresponding ones of the right sequence, and the right sequence is longer than or equal to the left. For this case, as P is a smaller pattern with a shorter left sequence and the same right sequence as Q, it must be the case that all elements of P's left sequence match with the corresponding ones of P's right sequence and that the right sequence is longer than the left. If this is the case, P is canonical, and hence a contradiction.

Part 2: Right-Extension. Consider for contradiction that a canonical right-extension pattern Q is grown from a non-canonical right-extension pattern P. There are only two conditions where a pattern Q is considered canonical.

The first condition requires the existence of an element x in the left sequence of Q to be smaller than the corresponding one in the right sequence, and all other previous elements are the same. As Q is grown from P, P must be a smaller pattern. As P and Q are right extension patterns, they must have the same left sequence. Thus there are again two cases. The first case is such that the length of the right sequence of P is longer than the element x, if this is the case P is canonical and thus a contradiction. The second case is such that the right sequence of P is shorter than the position of element x. For this to happen the right sequence must be shorter than the left. This is an impossible condition as for all right extension patterns the right sequence is longer than the left. Thus, there is a contradiction.

The second condition requires that all elements of the left sequence of Q to match with corresponding ones of the right sequence, and the right sequence is longer than the left. There are two cases. The first case is such that P's right sequence is longer or equal than the left. If this is the case than P is also canonical which is a contradiction. The second case is such that P's right sequence is shorter than the left, but this is impossible for right extension patterns. Thus, it is a contradiction. □

Summary. Combining the above points, our strategy to visit all the patterns in the search space is as follows:

1. Grow left-extension patterns leftwards (i.e., by only appending the left sequence) and grow right-extension patterns in both directions to prevent the same pattern from being generated multiple times.
2. Only generate canonical patterns.
3. We do not need to grow non-canonical patterns further.

In the following sections we elaborate on how we incorporate our search space traversal strategy with an effective data structure and search space pruning strategies into a holistic mining algorithm.

5. TANDEM PROJECTED DATABASE

To speed up the mining process, we propose a new data structure referred to as *tandem projected database*. This data structure extends the projected database first proposed in PrefixSpan to mine standard sequential patterns [20]. Similar to the original use of the projected database in [20], our tandem projected database is used to facilitate effective, fast, and incremental computation of pattern statistics, i.e., support and discriminative score. We refer to the operation of creating a tandem projected database as *tandem projection*. The reason of using the word "tandem" would be made clear in the following paragraphs.

Tandem projection operation takes in a database which is a set of sequence pairs and a dyadic sequential pattern. It then outputs suffixes of the pairs of sequences in the database that have the pattern as its prefix. Formally, we define it in Definition 5.1.

DEFINITION 5.1 (**Tandem Projection**). *Consider a database DB and a pattern $P = p1 - p2$. The tandem projection of DB with respect to P is defined as:*

$$\begin{aligned}
\{[(a,b),(c,d)] \mid \quad & s = s1 - s2 \in DB \land \\
& (s1 = pre_1{+\!+}a) \land (s2 = pre_2{+\!+}b) \land \\
& (s2 = pre_3{+\!+}c) \land (s1 = pre_4{+\!+}d) \land \\
pre_1 = & \; shortest\ prefix\ of\ s1\ containing\ p1\ or, \\
& \; if\ no\ such\ prefix\ exists, s1 \land \\
pre_2 = & \; shortest\ prefix\ of\ s2\ containing\ p2\ or, \\
& \; if\ no\ such\ prefix\ exists, s2 \land \\
pre_3 = & \; shortest\ prefix\ of\ s2\ containing\ p1\ or, \\
& \; if\ no\ such\ prefix\ exists, s2 \land \\
pre_4 = & \; shortest\ prefix\ of\ s1\ containing\ p2\ or, \\
& \; if\ no\ such\ prefix\ exists, s1 \land \\
& (a \neq \epsilon \land b \neq \epsilon) \lor (c \neq \epsilon \land d \neq \epsilon)\}
\end{aligned}$$

The resultant tandem projected database of database DB with respect to a pattern P is denoted as DB_P^{tnd}.

Each *tandem projected database entry* denoted as $[(a,b), (c,d)]$ in Definition 5.1 is composed of two elements (a,b) and (c,d). These are two *simple projected database entries* corresponding to a single sequence pair in the database. As there are two sequences in a sequence pair in the database, it is possible to map the left sequence of the pattern to the left sequence of the sequence pair, or alternatively, to the right sequence of the pair. The first one captures the left to left and right to right mapping. The other one captures the left to right and right to left mapping. We refer to (a,b) and (c,d) as the first and second simple projected database entry, respectively. We denote them as $DB_P^{tnd}.First$ and $DB_P^{tnd}.Second$. We refer to a in (a,b) as $DB_P^{tnd}.First.Left$; b, c, d are denoted in a similar fashion.

Each simple projected database entry captures the two suffixes of the corresponding sequence pair whose prefixes are the shortest prefixes containing the pattern's two sequences (lines 2-11 in Definition 5.1).

Note that for a sequence pair it is possible that only either one of (a,b) or (c,d) exists. If this is the case, we assign (ϵ,ϵ) to either (a,b) or (c,d) correspondingly. As we do not want to capture empty projected database entries, we restrict that either (a,b) or (c,d) must contain non-epsilon values (the last line in Definition 5.1).

Each *simple projected database entry* of a pattern $p1$-$p2$ in a sequence pair s in a database DB is represented by (i, l, r, idx_l, idx_r, alt), with the symbols defined as:

i	Index of s in DB.
l	Index of the sequence in the pair s mapped to $p1$
r	Index of the sequence in the pair s mapped to $p2$
idx_l	Starting index of the corresponding suffix of $s[l]$
idx_r	Starting index of the corresponding suffix of $s[r]$
alt	Pointer to another simple projected database entry in the tandem (if any)

The last element *alt* contains a pointer to another *simple projected database entry* which is a fellow member of the corresponding *tandem projected database entry*.

A tandem projected database is thus a set of all such tuples representing the simple projected database entries. Related simple projected database entries are tied together in tandem via the *alt* element of the tuple.

Example. Consider the database shown in Table 1. The *tandem projected database entry* of the first sequence pair with respect to pattern $\langle a, d\rangle$ - $\langle c, d\rangle$ is [($\langle d\rangle$,$\langle d, e\rangle$), (ϵ, ϵ)]. The first *simple projected database entry* in the tandem is internally represented as (1, 1, 2, 4, 4, *null*), where the first '1' denotes the index of the first sequence pair in the database, the second '1' denotes that the left sequence of the pattern is mapped to the left sequence of the sequence pair (index = 1), the '2' denotes that the right sequence of the pattern is mapped to the right sequence of the sequence pair (index = 2), the first '4' denotes the index of the suffix $\langle d\rangle$, and the last '4' denotes the index of the suffix $\langle d, e\rangle$. As the second *simple projected database entry* in the tandem is (ϵ, ϵ), the *alt* element of the tuple is set to null.

6. PROPERTIES & THEOREMS

In this section, we present some pruning properties to prune the uninteresting (i.e., infrequent, non-closed, or non-discriminative) patterns *en masse*. First, we describe the anti-monotonicity property of support in Property 2. With this property, we do not need to grow an infrequent pattern further as all its descendants would not be frequent either.

PROPERTY 2 (**Anti-Monotonicity of Support**). *The support of a pattern P is always greater than or equal to the support of its descendants.*

PROOF. *Without any loss of generality, consider an arbitrary descendant Q of P and an arbitrary sequence pair S in a database DB. Since Q is grown from P, $P \sqsubseteq_p Q$. If Q occurs in S, it must be the case that $Q \sqsubseteq_p S$. It is clear that the $\sqsubseteq_p$ operator is transitive. Thus since $P \sqsubseteq_p Q$, if $Q \sqsubseteq_p S$, it must be the case that $P \sqsubseteq_p S$, i.e., P occurs in the sequence pair S. Thus for any arbitrary sequence S where Q occurs, P also occurs in S. Thus the support of P must always be greater than or equal to the support of Q. Thus the above property holds.* $\Box$

Next we describe two properties related to a pattern's discriminative score in Properties 3 & 4. From these two properties, it is clear that if $disc_{ub}(P) < min_disc$, there is no need to grow P further. This is the case as the discriminative score upper bound of all descendants of P would be no larger than $disc_{ub}(P)$, which in turn is smaller than the *min_disc* threshold. Thus, all descendants of P are not discriminative.

PROPERTY 3 (**Upper Bound of Discrimin. Score**). *For pattern P and database DB, disc(P,DB) is bounded by:*

$$disc_{ub}(P) = \max(IG(sup_{+ve}(P), 0), IG(0, sup_{-ve}(P)))$$

We denote the upper bound on the discriminative score of a pattern P as $disc_{ub}(P)$.

PROOF. *Consider the discriminative score defined in Definition 3.9. Let $p = sup_{+ve}(P)$ and $q = sup_{-ve}(P)$. If we take a partial derivative wrt. p and q respectively, we have*

$$\frac{\partial IG(p,q)}{\partial p} = \frac{1}{|DB|} \log \frac{p(|DB| - (p+q))}{(p+q)(DB.Pos - p)},$$
$$\frac{\partial IG(p,q)}{\partial q} = \frac{1}{|DB|} \log \frac{q(|DB| - (p+q))}{(p+q)(DB.Neg - q)}.$$

$$\text{if } \frac{p}{DB.Pos} > \frac{q}{DB.Neg}, \frac{\partial IG(p,q)}{\partial p} > 0, \frac{\partial IG(p,q)}{\partial q} < 0;$$
$$\text{if } \frac{p}{DB.Pos} < \frac{q}{DB.Neg}, \frac{\partial IG(p,q)}{\partial p} < 0, \frac{\partial IG(p,q)}{\partial q} > 0.$$

Then we have the following conclusions:

$$\text{if } \frac{p}{DB.Pos} > \frac{q}{DB.Neg}, IG(p,q) \leq IG(p,0);$$
$$\text{if } \frac{p}{DB.Pos} < \frac{q}{DB.Neg}, IG(p,q) \leq IG(0,q).$$

Therefore, we prove

$$\begin{aligned} disc_{ub}(P) &= \max(IG(p,0), IG(0,q)) \\ &= \max(IG(sup_{+ve}(P), 0), IG(0, sup_{-ve}(P))) \end{aligned}$$

$\Box$

PROPERTY 4 (**Anti-Monotonicity of Disc. Bound**). *For pattern P and its descendant P', $disc_{ub}(P) \geq disc_{ub}(P')$.*

PROOF. *Let $p = sup_{+ve}(P)$ and $q = sup_{-ve}(P)$; $p' = sup_{+ve}(P')$ and $q' = sup_{-ve}(P')$. Since P' is a descendant of P, we have $p' \leq p$ and $q' \leq q$. According to Property 3, for the patterns P and P' we have:*

$$disc_{ub}(P) = \max(IG(p,0), IG(0,q))$$

and

$$disc_{ub}(P') = \max(IG(p',0), IG(0,q'))$$

According to the proof in Property 3, if $\frac{p}{DB.Pos} > \frac{q}{DB.Neg}$, $\frac{\partial IG(p,q)}{\partial p} > 0$*, we can conclude*

$$IG(p,0) \geq IG(p',0)$$

as $p \geq p'$ and $q = 0$ here. Similarly, we have

$$IG(0,q) \geq IG(0,q')$$

Without loss of generality, assume $IG(p',0) \geq IG(0,q')$. Then

$$disc_{ub}(P') = IG(p',0) \leq IG(p,0) \leq \max(IG(p,0), IG(0,q))$$

Therefore, we prove $disc_{ub}(P) \geq disc_{ub}(P')$. $\Box$

Before we present a property that could be used to cut the search space containing non-closed patterns, we first need to define some terms. In Definitions 6.1 & 6.2, we define some concepts pertaining to the occurrences of a sequence in another sequence. We present the concepts of *in-between event sets* and *strict in-between event sets* in Definitions 6.3 & 6.4 respectively.

DEFINITION 6.1 (**First instance**). *Given a sequence S containing a sequence $P = \langle e_1, \ldots e_n\rangle$. The first instance of P in S is the shortest prefix pre of S where $P \sqsubseteq pre$.*

DEFINITION 6.2 (**Last instance**). *Given a sequence S containing a sequence $P = \langle e_1, \ldots e_n\rangle$. The last instance of P in S is the prefix of S ending with the last occurrence of e_n in S.*

Example. Consider a sequence $S = \langle a, b, b, b, c, b, c, d\rangle$ and another sequence $P = \langle a, b\rangle$. The first instance of P in S is the prefix $\langle a, b\rangle$. The last instance of P in S is the prefix $\langle a, b, b, b, c, b\rangle$.

Definition 6.3 (**In-Between Event Sets**). *Given a pattern P=p1-p2, where* $p1 = \langle a_1, \ldots, a_n\rangle$ *and* $p2 = \langle b_1, \ldots, b_n\rangle$*, and a sequence pair S=s1-s2, there are* $|p1| + |p2|$ *in-between event sets:* $L_1, \ldots, L_{|p1|}$, $R_1, \ldots, R_{|p2|}$*, of P in S.*

The L_i *in-between event set is defined as:*

(If i = 1) Let $L_{1_{SL}}$ *be the set of events occurring before the first occurrence of* a_1 *in S.Left iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and* $\{\}$ *otherwise. Also, let* $L_{1_{SR}}$ *be the set of events occurring before the first occurrence of* a_1 *in S.Right iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and* $\{\}$ *otherwise. Then,* $L_1 = L_{1_{SL}} \cup L_{1_{SR}}$.

(If i > 1) Let $L_{i_{SL}}$ *be the set of events occurring between the first instance of* $\langle a_1 \ldots a_{i-1}\rangle$ *in S.Left, to the last occurrence of* a_i *in the last instance of* p1 *in S.Left iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and* $\{\}$ *otherwise. Also, let* $L_{i_{SR}}$ *be the set of events occurring between the first instance of* $\langle a_1 \ldots a_{i-1}\rangle$ *in S.Right, to the last occurrence of* a_i *in the last instance of* p1 *in S.Right iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and* $\{\}$ *otherwise. Then,* $L_i = L_{i_{SL}} \cup L_{i_{SR}}$.

The R_i *in-between event set is defined as:*

(If i = 1) Let $R_{1_{SL}}$ *be the set of events occurring before the first occurrence of* b_1 *in S.Left iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and* $\{\}$ *otherwise. Also, let* $R_{1_{SR}}$ *be the set of events occurring before the first occurrence of* b_1 *in S.Right iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and* $\{\}$ *otherwise. Then,* $R_1 = R_{1_{SL}} \cup R_{1_{SR}}$

(If i > 1) Let $R_{i_{SL}}$ *be the set of events occurring between the first instance of* $\langle b_1 \ldots b_{i-1}\rangle$ *in S.Left, to the last occurrence of* b_i *in the last instance of* p1 *in S.Left iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and* $\{\}$ *otherwise. Also, let* $R_{i_{SR}}$ *be the set of events occurring between the first instance of* $\langle b_1 \ldots b_{i-1}\rangle$ *in S.Right, to the last occurrence of* b_i *in the last instance of* p2 *in S.Right iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and* $\{\}$ *otherwise. Then,* $R_i = R_{i_{SL}} \cup R_{i_{SR}}$

We denote the L_i *and* R_i *in-between event sets of a pattern P in sequence S as* $L_i(P, S)$ *and* $R_i(P, S)$*, respectively.*

Example. Consider a sequence pair $S = \langle a, b, b, b, c, b, c, d\rangle$ - $\langle a, c, b, b, c, d, x\rangle$ and a pattern $P = \langle b, c\rangle$ - $\langle x\rangle$. The sets $L_{1_{SL}}$(P,S), $L_{1_{SR}}$(P,S), and L_1(P,S) are $\{a\}$, $\{a, c\}$, and $\{a, c\}$ respectively. The sets $L_{2_{SL}}$(P,S), $L_{2_{SL}}$(P,S), and L_2(P,S) are $\{b, c\}$, $\{b\}$, and $\{b, c\}$ respectively. The sets $R_{1_{SL}}$(P,S), $R_{1_{SR}}$(P,S), and R_1(P,S) are $\{\}$, $\{a, b, c, d\}$, and $\{a, b, c, d\}$ respectively.

Definition 6.4 (**Strict In-Between Event Sets**). *Given a pattern P=p1-p2, where* $p1 = \langle a_1, \ldots, a_n\rangle$ *and* $p2 = \langle b_1, \ldots, b_n\rangle$*, and a sequence pair S=s1-s2, there are* $|p1| + |p2|$ *strict in-between event sets:* $LT_1, \ldots, LT_{|p1|}$, $RT_1, \ldots, RT_{|p2|}$*, of P in S.*

The LT_i *strict in-between event set is defined as:*

(If i = 1) Let $LT_{1_{SL}}$ *be the set of events occurring before the first occurrence of* a_1 *in S.Left iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and the alphabet* Δ *otherwise. Also, let* $LT_{1_{SR}}$ *be the set of events occurring before the first occurrence of* a_1 *in S.Right iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and the alphabet* Δ *otherwise. Then, if* $LT_1 = LT_{1_{SL}} \cap LT_{1_{SR}}$.

(If i > 1) Let $LT_{i_{SL}}$ *be the set of events occurring between the first instance of* $\langle a_1 \ldots a_{i-1}\rangle$ *in S.Left, to the last occurrence of* a_i *in the* first *instance of* p1 *in S.Left iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and the alphabet* Δ *otherwise. Also, let* $LT_{i_{SR}}$ *be the set of events occurring between the first instance of* $\langle a_1 \ldots a_{i-1}\rangle$ *in S.Right, to the last occurrence of* a_i *in the* first *instance of* p1 *in S.Right iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and the alphabet* Δ *otherwise. Then,* $LT_i = LT_{i_{SL}} \cap LT_{i_{SR}}$.

The RT_i *strict in-between event set is defined as:*

(If i = 1) Let $RT_{1_{SL}}$ *be the set of events occurring before the first occurrence of* b_1 *in S.Left iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and the alphabet* Δ *otherwise. Also, let* $RT_{1_{SR}}$ *be the set of events occurring before the first occurrence of* b_1 *in S.Right iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and the alphabet* Δ *otherwise. Then,* $RT_1 = RT_{1_{SL}} \cap RT_{1_{SR}}$

(If i > 1) Let $RT_{i_{SL}}$ *be the set of events occurring between the first instance of* $\langle b_1 \ldots b_{i-1}\rangle$ *in S.Left, to the last occurrence of* b_i *in the* first *instance of* p1 *in S.Left iff* $p1 \sqsubseteq s2$ *and* $p2 \sqsubseteq s1$*, and the alphabet* Δ *otherwise. Also, let* $RT_{i_{SR}}$ *be the set of events occurring between the first instance of* $\langle b_1 \ldots b_{i-1}\rangle$ *in S.Right, to the last occurrence of* b_i *in the* first *instance of* p2 *in S.Right iff* $p1 \sqsubseteq s1$ *and* $p2 \sqsubseteq s2$*, and the alphabet* Δ *otherwise. Then,* $RT_i = RT_{i_{SL}} \cap RT_{i_{SR}}$.

We denote the LT_i *and* RT_i *strict in-between event sets of a pattern P in sequence S as* $LT_i(P, S)$ *and* $RT_i(P, S)$*, respectively.*

Example. Consider a sequence pair $S = \langle a, b, b, b, c, b, c, d\rangle$ - $\langle a, c, b, b, c, d, x\rangle$ and a pattern $P = \langle b, c\rangle$ - $\langle x\rangle$. The sets $LT_{1_{SL}}$(P,S), $LT_{1_{SR}}$(P,S), and LT_1(P,S) are $\{a\}$, $\{a, c\}$, and $\{a\}$ respectively. The sets $LT_{2_{SL}}$(P,S), $LT_{2_{SL}}$(P,S), and LT_2(P,S) are $\{b\}$, $\{b\}$, and $\{b\}$ respectively. The sets $RT_{1_{SL}}$ (P,S), $RT_{1_{SR}}$(P,S), and RT_1(P,S) are Δ, $\{a, b, c, d\}$, and $\{a, b, c, d\}$ respectively.

Definitions 6.3 & 6.4 capture events that appear between the occurrences of two events (or before the occurrence of the first event) of a pattern in a sequence pair containing it. From the definition, we could notice that a strict in-between event set is a subset of the corresponding in-between event set. Remember that there are two different ways to match a pattern to a sequence pair: left to left and right to right, or left to right and right to left. Also, there could be multiple occurrences of a pattern in a sequence. An in-between event set captures events that occur in between two pattern's events for at least *one* possible occurrence of the pattern in the sequence in at least *one* possible matching. A strict in-between event set captures events that occur in between two pattern's events for *all* possible occurrences of the pattern in the sequence in *both* possible matchings (if they exist)[1].

We next define the concept of forward extension and backward extension in Definitions 6.5 & 6.6. Two properties specifying how these two sets can be computed are given in Properties 5 & 6.

Definition 6.5 (**Forward Extension**). *A forward extension event of a pattern P is an event that could be appended to P (i.e., any sequence of P) resulting in another pattern with the same support.*

Example. Consider the example database ExDB shown in Table 1 and a pattern $P = \langle a\rangle$ - $\langle d\rangle$. Event d is a forward extension event of P as there is a pattern $Q = \langle a, d\rangle$ - $\langle d\rangle$ that could be formed from P by appending the event d at the end of the left sequence of P and $\sup(P) = \sup(Q) = 8$.

Property 5 (**Forward Extension Set**). *The forward*

[1]Notice that Definition 6.3 uses the last occurrence in the first instance, $\{\}$, and set union operator while Definition 6.4 uses the last occurrence in the last instance, Δ, and set intersection operator.

extension event set of a pattern P is the set:

$$\{e|\forall[ef, es] \in DB_P^{tnd}.(e \in ef.Left \vee e \in es.Left \vee e \in ef.Right \vee e \in es.Right)\}$$

Proof.

Part 1: If e is a forward extension event, e is in the set. *Since e is appended to P, e must be in the projected database of P. Since the support of P appended by e is the same as P, e must appear in all tandem projected database entries of P; e could appear either in the first or second simple projected database entries in the tandem (either in the left sequence or the right sequence).*

Part 2: If e is in the set, e is a forward extension event. *If e is in the set, e appears in all tandem projected database entries (either in the first or the second entry in each tandem entry, either in the left or in the right sequence). Thus we could extend P with e (either in the left or right sequence) resulting in another pattern Q with no less support than P. From the anti-monotonicity property, Q could not have a higher support. Thus, sup(P) = sup(Q).* □

Definition 6.6 (**Backward Extension**). *A backward extension of a pattern P is an event that could be inserted to P (i.e., any sequence of P) resulting in another pattern with the same support.*

Example. Consider the example database ExDB shown in Table 1 and a pattern $P = \langle a, d\rangle$ - $\langle e, c, d, d, e\rangle$. Event b is a backward extension event of P as there is a pattern Q = $\langle a, b, d\rangle$ - $\langle e, c, d, d, e\rangle$ that could be formed from P by inserting the event b to the left sequence of P and sup(P) = sup(Q) = 5.

Property 6 (**Backward Extension Set**). *The backward extension set of a pattern P are events appearing in one of the in-between event sets of P in all sequence pairs supporting P in the database. Mathematically, this is the set:*

$$\{e|\exists x \in \{L_1, \ldots, L_{|P.Left|}, R_1, \ldots, R_{|P.Right|}\}. \forall_{(S\in DB)\wedge(P\sqsubseteq S)}.e \in x(P, S)\}$$

Proof.

Part 1: If e is a backward extension event, e is in the set. *Since e is a backward extension event, it must be inserted between one of the events of the pattern or before the first event of the pattern. This together with the fact that the support after the insertion of e is the same as the original support guarantee that e must appear in the same in-between event sets for all sequence pairs in the database supporting P.*

Part 2: If e is in the set, e is a backward extension event. *If e is in the set, for all sequence pair S supporting P, it is possible to insert event e such that the resulting pattern Q is a super-sequence of P and is a subsequence of S, i.e., $P \sqsubseteq_p Q \sqsubseteq_p S$. Thus the support of the resultant pattern Q is the same as P, and thus e is a backward extension event.* □

Next, Property 7 describes a relationship between the support and the discriminative score of two related patterns. From the property, it could be inferred that appending or inserting backward and forward extension events at the corresponding locations would result in patterns not only having the same support, but also the same discriminative score.

Property 7 (**Support and Disc. Score**). *Consider two patterns P and Q where $P \sqsubseteq_p Q$. If sup(P) = sup(Q), then disc(P) = disc(Q).*

Proof. *Without any loss of generality, consider an arbitrary sequence pair S in the input database. If $Q \sqsubseteq_p S$, it must be the case that $P \sqsubseteq_p S$ too. Thus any sequence pair supporting Q will also support P. If furthermore, sup(P) = sup(Q), it must be the case that the patterns P and Q are supported by the same set of sequence pairs. Thus disc(P) must be equal to disc(Q).* □

Based on the above definitions and properties, Property 8 specifies a way to check if a pattern is closed. This check is useful as we could use it to output a closed pattern right away instead of comparing it with the set of all frequent and discriminative patterns to decide whether it is closed or not. Next, Property 9 defines a way to prune the search space containing non-closed patterns.

Property 8 (**Closure Check**). *If a pattern has no forward extension and no backward extension, then it is closed.*

Proof. *A pattern can only be grown to another pattern either by appending an event, or inserting an event. From the anti-monotonicity of pattern support, if appending or inserting an event reduces the support of a pattern, appending or inserting more events would not increase the support anymore. Thus, if there is no event either being inserted (i.e., backward extension) or appended (i.e., forward extension) to result in another pattern having the same support, the pattern must be closed.* □

Example. Consider the example database ExDB shown in Table 1 and a pattern $P = \langle a, b, d, d\rangle$ - $\langle e, c, d, d, e\rangle$. Since P has no forward and no backward extension, it is a closed pattern.

Property 9 (**Non-Closedness Pruning**). *If there is an event in one of P strict in-between event sets for all sequences containing P in DB, then P and all descendants of P are not closed.*

Proof. *If there exists such an event e, it means for every sequence pair containing P in the database, it is possible to insert e in P resulting in another pattern Q such that $DB_P = DB_Q$. Thus, any extensions that could be appended to P resulting in a pattern R could also be appended to Q resulting in a pattern R' such that sup(R) = sup(R'). From Property 7, it must also be the case that disc(R) = disc(R'). Thus, P and its descendants are not closed.* □

Example. Consider the example database ExDB shown in Table 1 and a pattern $P = \langle a\rangle$ - $\langle e, c, e\rangle$. For all sequence S containing P, there is an event d in $RT_3(P, S)$, thus P and all descendants of P are not closed.

7. MINING ALGORITHMS

We describe three algorithm variants to mine discriminative dyadic sequential patterns: `baseline`, `all-frequent`, and `closed`. The `baseline` variant is a simple approach that does not utilize much pruning properties and data structure. The `all-frequent` variant employs a number of pruning properties to mine all frequent and discriminative dyadic sequential patterns. The `closed` variant employs pruning properties to remove non-closed dyadic sequential patterns early to result in a closed set of discriminative and frequent dyadic sequential patterns.

7.1 Baseline Algorithm Variant

In this variant, we employ a simple approach to mine discriminative and frequent dyadic sequential patterns. The steps are:

1 Combine the left and right sequences of the pairs in the database into one sequence database SDB.
2 Mine frequent sequential patterns with support no less than the user-defined *min_sup* from SDB.
3 Pair every frequent sequential pattern to build a set of candidate dyadic patterns $Cand$.
4 For every candidate pattern in $Cand$, test its support and discriminative score. Only output patterns that satisfy the minimum support and discriminative threshold, *min_sup* and *min_disc* respectively.

The approach simply decomposes the problem to standard sequential pattern mining, then re-assembling mined patterns to form discriminative and frequent dyadic sequential patterns. Note that we need to mine all frequent standard sequential patterns – closed standard sequential patterns could not be employed – as otherwise some dyadic sequential patterns would be lost.

7.2 Mining All Frequent and Discriminative Patterns

In this variant, we would like to mine all frequent and discriminative patterns. We make use of the search space traversal strategy described in Section 4 and the dyadic projected database described in Section 5. Several pruning properties described in Section 6 would also be utilized. The pseudocode is shown in Figure 3.

We start with patterns of size 2, i.e., those composed of two sequences each having one event, that satisfy the *min_sup* and *min_disc* thresholds (line 1). From this base set of patterns, we grow it leftwards and rightwards following the procedure in Section 4 (lines 3-5). We only grow patterns that are canonical following Property 1 (lines 1, 12, and 20). For every pattern that is generated, its tandem project database is computed and is used to quickly compute the tandem projected database of its immediate descendants (lines 2, 13, and 21). Thus, partial result when computing the support of a dyadic pattern could be used to help compute the support of its descendants.

The anti-monotonicity of support, i.e., Property 2 is used to prune the search space of infrequent patterns en-masse (Lines 1, 14, and 22). If a pattern is not frequent there is no need to grow the pattern further. Indeed all descendants of the pattern would also be not frequent. Similarly the upper bound of the discriminative score and the anti-monotonicity of this upper bound, i.e., Properties 3 & 4 are used to prune the search space of non discriminative patterns en-masse (lines 1, 14, and 22). If a pattern's discriminative score upper bound does not satisfy the *min_disc* threshold, we could stop growing the pattern further, as all extensions of the pattern would not be discriminative either.

7.3 Mining Closed Discriminative Patterns

In this variant, we push the early elimination of non-closed patterns into the search space traversal. In particular, we use Property 9 to prune the search space of non-closed patterns en-masse. We also use Property 8 to immediately check if a pattern is closed. Thus, no intermediate patterns need to be stored in memory. Any patterns that are detected as closed and discriminative could directly be output.

Procedure MineAllFrequent
Inputs:
DB : Database of sequence pairs
min_sup : Minimum support threshold
min_disc: Minimum discriminative threshold
Output:
All patterns that are frequent and discriminative
Methods:
1: Let Base = {p=p1-p2|sup(p) $\geq$ *min_sup* $\wedge$ |p1|=|p2|=1 $\wedge$ disc$_{ub}(p) \geq$ *min_disc* $\wedge$ p is canonical}
2: Compute tandem projected databases for all patterns in Base wrt. DB
3: For each p in Base
4: Grow(p, "L", *min_sup*, *min_disc*)
5: Grow(p, "R", *min_sup*, *min_disc*)

Procedure Grow
Inputs:
$p = p1\text{-}p2$: Pattern to be grown
Dir : Direction of extension ("L" or "R")
min_sup = Minimum support threshold
min_disc = Minimum discriminative threshold
Output:
Extended patterns that are discriminative and frequent
Methods:
6: If (disc(p) $\geq$ *min_disc*)
7: Output p
8: Let PDB = projected database of p
// Grow Left
9: Let LFE_L = {ev|exists $\geq$ n entries [(a,b),(c,d)] in PDB with ev $\in$ a or $ev \in$ c}
10: For each event e_L in LFE_L
11: Let p' = ($p1$++ e_L)-$p2$
12: If p' is canonical
13: Compute projected database of p' from PDB
14: If (sup(p') $\geq$ *min_sup* $\wedge$ disc$_{ub}$(p') $\geq$ *min_disc*)
15: Grow(p', "L", *min_sup*, *min_disc*)
// Grow Right
16: If (Dir="R")
17: LFE_R = {ev|exists $\geq$ n entries [(a,b),(c,d)] in PDB with ev $\in$ b or ev $\in$ d}
18: For each event e_R in LFE_R
19: Let p' = $p1$-($p2$++e_R)
20: If p' is canonical
21: Compute projected database of p' from PDB
22: If (sup(p') $\geq$ *min_sup* $\wedge$ disc$_{ub}$(p') $\geq$ *min_disc*)
23: Grow(p', "R", *min_sup*, *min_disc*)

Figure 3: Mining All Frequent Discriminative Patterns

The pseudocode of the `closed` variant is similar to the one shown in Figure 3, with a few differences related to the two points mentioned in the proceeding paragraph. In Figure 4, we only highlight several locations that are changed. Note that we add non-closedness pruning checks following Property 9 at lines c1 and c2. Before outputing a pattern we also perform the closure check based on Property 8 at line 6.

8. EXPERIMENTS

In this section, we first describe our experimental setting. We then present the results of our scalability experiments followed by our case study on duplicate bug reports.

8.1 Experimental Settings & Datasets

Experimental Settings. All experiments are performed on an Intel Xeon E5540 2.53GHz server with 24.0 GB of RAM running Windows Server 2008 R2 Standard (64 bit). Algorithms are written in Visual C#.Net.

Synthetic Datasets. Our synthetic data generation con-

Procedure MineClosed
Output:
All patterns that are closed, frequent, and discriminative
...

Procedure Grow
Output:
Extended pat. that are closed, discriminative, and frequent
Methods:

```
6:  If (disc(p) ≥ min_disc ∧ p satisfies Property 8)
7:     Output p
...
// Grow Left
...
14:       If (sup(p') ≥ min_sup ∧ disc_ub(p') ≥ min_disc)
c1:          If (p' is not prunable by Property 9)
15:             Grow(p', "L", min_sup, min_disc)
// Grow Right
...
22:       If (sup(p') ≥ min_sup ∧ disc_ub(p') ≥ min_disc)
c2:          If (p' is not prunable by Property 9)
23:             Grow(p', "R", min_sup, min_disc)
```

Figure 4: Mining Closed Discriminative Patterns

sists of two steps: labeled sequence generation step, and sequence pairings step. We build our synthetic data generator on top of the IBM Data Generator [5].

Labeled Sequence Pair Generation Step. We modify the IBM data generator to generate sequences of events rather than sequences of sets of events. We set all parameters of the IBM Data Generator to its default value except for the number of sequences D. We randomly assign each sequence to one of the two classes with equal likelihood. We split each sequence into a pair of sequences of approximately equal length (i.e., the lengths differ by one if the original sequence has an odd number of events).

Discriminative Pattern Insertion. Next, we randomly create discriminative patterns with size (i.e., the number of events in the pattern) randomly chosen between 2 and $PSize$. We created $PNum$ patterns. We inject half of them into $PInst$ randomly chosen sequence pairs with +ve labels. The other half is injected into $PInst$ randomly chosen sequence pairs with -ve labels. At the end of this step, we have two sets of labeled sequence pairs.

Real Dataset. For the real dataset, we use bug reports from the bug repositories of three large open source projects: the Eclipse project [1], the Mozilla Firefox project [2] and the OpenOffice project [3]. Eclipse is a popular open source integrated development environment written in Java; Firefox is a well-known open source web browser written in C/C++, and OpenOffice is an open source counterpart of Microsoft Office. We use the bug reports submitted to OpenOffice in year 2008 (12,732 bug reports), the bug report submitted to Eclipse in year 2008 (44,652 bug reports), and the bug reports submitted to Firefox from mid 2002 to mid 2007 (47,704 bug reports).

The bug repositories contain information on duplicated pairs of bug reports. The older report is called master, while the newer report is called duplicate. For each pair of duplicated bug reports, we form a sequence pair by the following process[2]:

1. Extract the word tokens from the summary field of the master and duplicate bug reports.
2. Perform stemming and stop word removal on the word tokens.
3. Convert the two sequences of word tokens into two sequences of integers by mapping a token to an integer. We ended up with a sequence pair.
4. Assign class label +ve to these sequence pairs.

To create the negative sequence pairs, we perform the following steps:

1. For each bug report marked as *duplicate*, randomly find another *master* bug report which is not its *master*.
2. Perform a similar list of steps as done for the pair of duplicated bug reports to get a pair of sequences.
3. Assign class label -ve to these sequence pairs.

We perform the steps mentioned above on the three sets of bug reports and combine the resultant sequence pairs. This dataset contains 11,898 pairs (5,949 +ve, and 5,949 -ve) composed from 8,601 different events. The pairs have on average 13.75 events. The largest pair has 62 events.

8.2 Scalability Experiments

We investigate the effect of changing various parameters, including min_sup, number of sequences D, number of injected patterns $PNum$, and maximum size of injected patterns $PSize$. We fix the values of the other parameters: $min_disc = 0.0001$ (to test scalability), and $PInst = 100$.

Varying the Minimum Support Threshold. The experiment results for varying support is shown in Figures 5–7. We plot the runtime and number of mined patterns that are generated on three configurations: a synthetic dataset with 10,000 sequences, a synthetic dataset with 25,000 sequences, and the real bug report dataset respectively. We vary the support threshold from 20 to 100 sequences.

From Figure 5, we notice that mining closed patterns could be up to two degrees of magnitude (more than 300 times) faster than mining all frequent and discriminative patterns. The number of all frequent and discriminative patterns could be more than 15,000 times larger than the number of closed ones. Compared to the number of injected patterns, even the `closed` variant mines more patterns as new frequent and discriminative patterns could be introduced by injecting this pattern on the original sequence pair dataset built using IBM data generator. The `basic` variant is not able to complete on any of the support thresholds considered after running for more than 8 hours. When we reduce the support threshold, the runtime and number of mined patterns increase for both `closed` and `all` variants. The absolute amounts of increment in the runtime and number of mined patterns are higher for `all` than `closed` variants. Note the graphs are plotted in log scale. Also, we inject frequent discriminative patterns in 100 sequences; thus mining at support level lower than 100 only adds "background" patterns. For these "background" patterns, many of the frequent and discriminative ones are not subsumed by a superpattern with the same support and discriminative score.

From Figure 6, we notice that only `closed` configuration is able to complete. The `basic` and `all` variants are not able to complete after running for more than 8 hours.

From Figure 7, we notice that on the real dataset the `closed` variant could complete on all thresholds even on a very low support threshold of 2 sequence pairs. For the real dataset, we need to run with low thresholds as the dataset of software problems expressed in English is sparse: there

[2] In this study, we do not generate a sequence pair for duplicate bug reports having the same master. This could also be done.

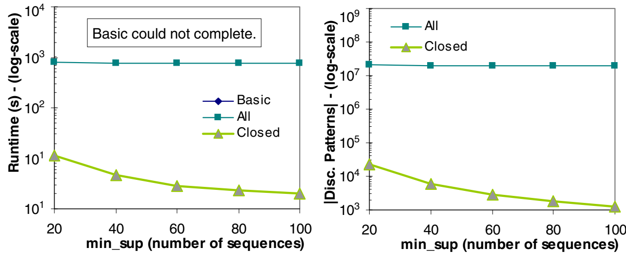

Figure 5: Varying the minimum support threshold min_sup **with** D **= 10k,** $PNum$ **= 10, and** $PSize$ **= 30: runtime (left), and number of patterns (right)**

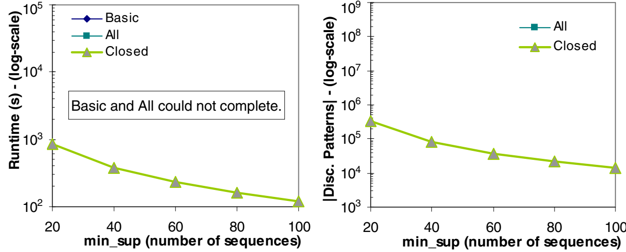

Figure 6: Varying the minimum support threshold min_sup **with** D **= 25k,** $PNum$ **= 30, and** $PSize$ **= 30: runtime (left), and number of patterns (right)**

are many problems and each problem could be expressed using a variety of word sequences. The `all` variant could complete on all thresholds except $min_sup = 2$. The differences in runtime and number of mined patterns between `all` and `closed` increase as the support threshold is reduced. At $min_sup = 4$, `all` variant is 131 times slower than the `closed` variant; also, `all` variant mines 1681 times more patterns (a total of 38,475,467 patterns are mined) than the `closed` variant. The `basic` variant is not able to complete after running for more than 8 hours.

Varying the Number of Sequences. Figure 8 plots the runtime needed and the number of patterns mined when we increase the size of the sequence pair dataset from 5,000 to 25,000 sequences. In general, when we increase the number of sequences, the runtime increases. There is not much increase in the number of patterns as we injected the same set of patterns. Only minor variations to the number of patterns exist due to the injection of the patterns into the randomly generated sequence pair database. A higher cost is incurred when mining a similar amount of patterns from a larger database as more events would need to be traversed.

Varying the Number of Injected Patterns. Figure 9 plots the runtime needed and the number of patterns mined when we increase the number of injected patterns from 10 to 50 sequences. We note that the runtime increases as there are more patterns to be mined.

Varying the Size of Injected Patterns. Figure 10 plots the runtime needed and the number of patterns mined when we increase the size of the injected patterns from 30 to 38. We note that the runtime and the number of patterns do not vary much. This shows the power of the closed pattern mining strategy that prunes the search space of non-closed patterns en-masse.

8.3 Case Study: Duplicate Bug Reports

In this section, we use mined patterns as features to detect whether two bug reports are duplicate of each other or not.

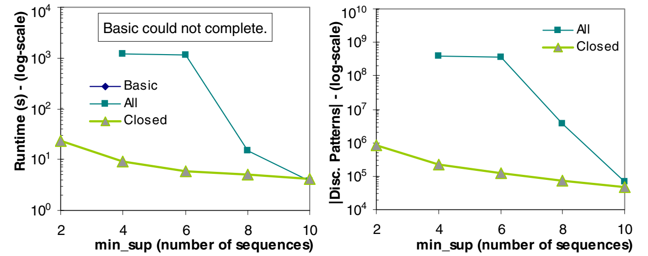

Figure 7: Varying the minimum support threshold min_sup **on the real dataset: runtime (left), and number of patterns (right)**

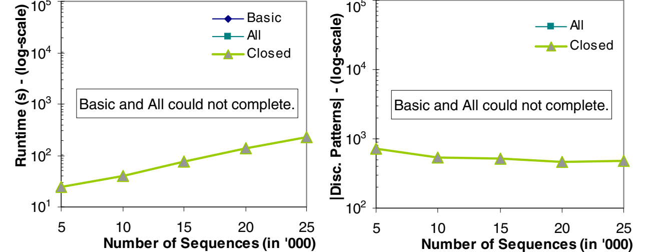

Figure 8: Varying the number of sequences D **with** min_sup **= 60,** $PNum$ **= 30, and** $PSize$ **= 30: runtime (left), and number of patterns (right)**

We use the same dataset as we use to mine patterns shown in Figure 7. We split the dataset into training and test. Given a training data containing pairs of bug reports, we extract features and learn a discriminative model.

We use LIBSVM [6] with probability estimates as the classification model. Classification accuracy, defined as the percentage of test cases correctly classified, is used as one measure. The measure AUC which is the area under a ROC curve is also used. ROC curve shows the trade-off between true positive rate and false positive rate for a given classifier [13]. A good classifier would produce a ROC curve as close to the top-left corner as possible. AUC is a measure of the model accuracy, in the range of $[0, 1.0]$. The best possible classifier would generate an AUC value of 1.0.

We consider three feature sets for LIBSVM classification:

1. Single Tokens. We use a vector corresponding to the set of tokens appearing in each pair of bug report.
2. Dyadic Patterns. we use a vector corresponding to the appearance/non-appearance of mined patterns in each pair of bug report. Patterns are mined from the training set with the minimum support and discriminative score thresholds set at 2 and 0.0001 respectively.
3. Both. We combine the above two feature sets.

We perform 10-fold cross validation, where for each we keep 1/10 of the data for testing and the other for training. We keep the class distribution for the test and training set to be (almost) equal. The results for the three feature sets in terms of accuracy and AUC are shown in Table 4.

The result shows that by using dyadic patterns as features we are able to classify pairs of bug reports accurately with more than 80% accuracy and 0.90 AUC. Using patterns to

Configuration	Accuracy	AUC
Single Tokens	60.38%	0.65
Dyadic Patterns	82.86%	0.90
Both	81.23%	0.89

Table 4: Accuracy: Duplicate Bug Report Detection

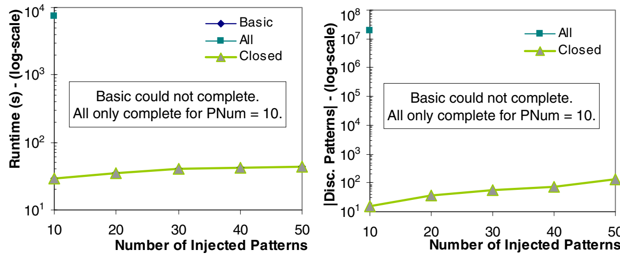

Figure 9: Varying number of patterns $PNum$ with min_sup = 60, D = 10k, and $PSize$ = 30: runtime (left), and number of patterns (right)

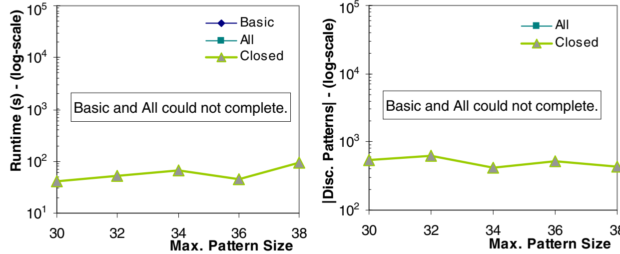

Figure 10: Varying maximum pattern size $PSize$ with min_sup = 60, D = 10k, and $PNum$ = 30: runtime (left), and number of patterns (right)

classify bug reports improves both accuracy and AUC by an addition of 22.48% and 0.35 respectively.

9. CONCLUSION AND FUTURE WORK

In this work, we propose a new problem of mining from a database of labeled sequence pairs. We are interested in mining dyadic sequential patterns that appear frequently in the database, and could discriminate between two classes, +ve and -ve. Rather than mining all possible patterns satisfying the above criteria, we only mine a compact representation of such patterns referred to as closed patterns.

To realize our solution, we propose a new search space traversal strategy, canonical representation of patterns, and a pruning strategy based on the canonical representation of patterns, which avoids the generation and test of many redundant patterns. We also propose a new projected database data structure to address the issue with having two sequences in a pattern causing two possible ways to project a pattern on a sequence pair. We propose a number of new search space pruning strategies that leverage anti-monotonicity properties on support and the upper bound of discriminative score. We also present a closure checking property that could detect closed patterns on the fly and a new property to eliminate non-closed patterns en-masse. We embed the above strategy, data structure, and properties in several algorithm variants.

We experiment on synthetic and real datasets to test the scalability and utility of our mining solution. We show that mining closed patterns could be much faster than mining all frequent and discriminative patterns. The latter could also be much faster than our baseline solution. On many settings, the `closed` variant can complete, while `all` and `baseline` can not. Patterns mined from the real software bug report dataset show that mined patterns could be used to detect duplicate bug reports.

As a future work, it is interesting to improve the scalability of the mining process further, experiment with more case studies (including cases where the two sequences in the pairs are asymmetric, i.e., representing different domains/concepts), and extend the study to mine even more expressive patterns, e.g., triadic sequential patterns, etc.

Acknowledgement. We would like to thank the anonymous reviewers for their valuable and constructive comments and advice. This work was supported in part by the Hong Kong Research Grants Council (RGC) General Research Fund (GRF) Project No. CUHK 411310.

10. REFERENCES

[1] Eclipse Bug Repository. https://bugs.eclipse.org/bugs/.
[2] Mozilla Bug Repository. https://bugzilla.mozilla.org/.
[3] OpenOffice Bug Repository. http://www.openoffice.org/issues/query.cgi.
[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.
[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, 1995.
[6] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/∼/cjlin/libsvm.
[7] H. Cheng, X. Yan, J. Han, and C. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, pages 716–725, 2007.
[8] H. Cheng, X. Yan, J. Han, and P. Yu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, 2008.
[9] S. Cong, J. Han, and D. Padua. Parallel mining of closed sequential patterns. In *KDD*, 2005.
[10] B. Ding, D. Lo, J. Han, and S.-C. Khoo. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *ICDE*, 2009.
[11] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD*, 1999.
[12] C. Gao, J. Wang, Y. He, and L. Zhou. Efficient mining of frequent sequence generators. In *WWW*, 2008.
[13] J. Han and M. Kamber. *Data Mining: Concepts and Techniques (2nd ed.)*. Morgan Kaufmann, 2006.
[14] W. S. Jr. and E. White. *The Elements of Style, 4th Ed.* Longman.
[15] D. Lo, H. Cheng, J. Han, S.-C. Khoo, and C. Sun. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *KDD*, 2009.
[16] D. Lo, S.-C. Khoo, and J. Li. Mining and ranking generators of sequential patterns. In *SDM*, 2008.
[17] D. Lo, S.-C. Khoo, and C. Liu. Efficient mining of iterative patterns for software specification discovery. In *Proc. 2007 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'07)*, pages 460–469, San Jose, CA, Aug. 2007.
[18] D. Lo, S.-C. Khoo, and L. Wong. Non-redundant sequential rules - theory and algorithm. *Information Systems*, 34(4-5):438–453, 2009.
[19] P. Nakov and H. Ng. Improved statistical machine translation for resource-poor languages using related resource-rich languages. In *EMNLP*, pages 1358–1367, 2009.
[20] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, 2001.
[21] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
[22] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *ICSE*, pages 45–54, 2010.
[23] T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, 2004.
[24] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *ICDE*, 2004.
[25] J. Wang, J. Han, and J. Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *KDD*, pages 236–245, 2003.
[26] X. Yan, H. Cheng, J. Han, and P. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, 2008.
[27] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In *SDM*, 2003.