Mining Asynchronous Periodic Patterns in Time Series Data

Jiong Yang
T.J. Watson Research Center
IBM
jiyang@us.ibm.com

Wei Wang
Department of Computer Science
University of North Carolina at Chapel Hill
weiwang@cs.unc.edu

Philip S. Yu
T.J. Watson Research Center
IBM
psyu@us.ibm.com

Abstract

Periodicy detection in time series data is a challenging problem of great importance in many applications. Most previous work focused on mining synchronous periodic patterns and did not recognize misaligned presence of a pattern due to the intervention of random noise. In this paper, we propose a more flexible model of asynchronous periodic pattern that may be present only within a subsequence and whose occurrences may be shifted due to disturbance. Two parameters min_rep and max_dis are employed to specify the minimum number of repetitions that is required within each segment of non-disrupted pattern occurrences and the maximum allowed disturbance between any two successive valid segments. Upon satisfying these two requirements, the longest valid subsequence of a pattern is returned. A two phase algorithm is devised to first generate potential periods by distance-based pruning followed by an iterative procedure to derive and validate candidate patterns and locate the longest valid subsequence. We also show that this algorithm can not only provide linear time complexity with respect to the length of the sequence but also achieve space efficiency.

Keywords

Asynchronous periodic pattern, Segment-based approach, Partial periodicity

1 Introduction

Periodicy detection on time series data is a challenging problem of great importance in many real applications. Most previous research in this area assumed that the disturbance within a series of repetitions of a pattern, if any, would not result in the loss of synchronization of subsequent occurrences of the pattern with previous occurrences [12] [13]. For example, "Joe Smith reads newspaper every morning" is a periodic pattern. Even though Joe might not read newspaper in the morning occasionally, this disturbance will not affect the fact that Joe reads newspaper in the morning of the subsequent days. In other words, disturbance is allowed only in terms of "missing occurrences" but not as general as any "insertion of random noise events". However, this assumption is often too restrictive since we may fail to detect some interesting pattern if some of its occurrences is misaligned due to inserted noise events. Consider the application of *inventory replenishment*. The history of inventory refill orders can be regarded as a symbol sequence. Assume that the time between two replenishments of cold medicine is a month normally. The refill order is filed at the beginning of each month before a major outbreak of flu which in turn causes an additional refill at the 3rd week. Afterwards, even though the replenishment frequency is back to once each month, the refill time shifts to the 3rd week of a month (not the beginning of the month any longer). Therefore, it would be desirable if the pattern can still be recognized when the disturbance is within some reasonable threshold.

In addition, the system behavior may change over time. Some pattern may not be present all the time (but rather within some time interval). Therefore, in this paper we aim at mining periodic patterns that are significant within a subsequence of symbols which may contain disturbance of length up to a certain threshold. Two parameters, namely *min_rep* and *max_dis*, are employed to qualify valid patterns and the symbol subsequence containing it, where this subsequence in turn can be viewed as a list of **valid segments** of perfect repetitions interleaved by disturbance. Each valid segment is required to be of at least min_rep contiguous repetitions of the pattern and the length of each piece of disturbance is allowed only up to max_dis. The intuition behind this is that a pattern needs to repeat itself at least a certain number of times to demonstrate its significance and periodicy. On the other hand, the disturbance between two valid segments has to be within some reasonable bound. Otherwise, it would be more appropriate to treat such disturbance as a signal of “change of system behavior” instead of random noise injected into some persistent behavior. The parameter max_dis acts as the boundary to separate these two phenomena. Obviously, the appropriate values of these two parameters are application dependent and need to be specified by the user. For patterns satisfying these two requirements, our model will return the subsequence with the maximum overall repetitions. Note that, due to the presence of disturbance, some subsequent valid segment may not be well synchronized with the previous ones. (Some position shifting occurs.) This in turn would impose a great challenge in the mining process.

Similar to [13], we allow a pattern to be partially filled to enable a more flexible model. For instance, (*cold_medi*, *, *, *) is a partial monthly pattern showing that the cold medicine is reordered on the first week of each month while the replenishment orders in the other three weeks do not have strong regularity. However, since we also allow the shifted occurrence of a valid segment, this flexible model poses a difficult problem to be solved. For a give pattern P, its associated valid segments may overlap. In order to find the valid subsequence with the most repetitions for P, we have to decide which valid segment and more specifically which portion of a valid segment should be selected. While it is relatively easy to find the set of valid segments for a given pattern, substantial difficulties lie on how to assemble these valid segments to form the longest valid subsequence. As shown in Figure 1, with $min\_rep = 3$, S_1, S_2, and S_3 are three valid segments of the pattern $P = (d_1, *, *)$. If we set $max\_dis = 3$, then X_1 is the longest subsequence before S_3 is considered, which in turn makes X_2 the longest one. If we only look at the symbol sequence up to position j without looking ahead in the sequence, it is very difficult to determine whether we should switch to S_2 to become X_1 or continue on S_1.

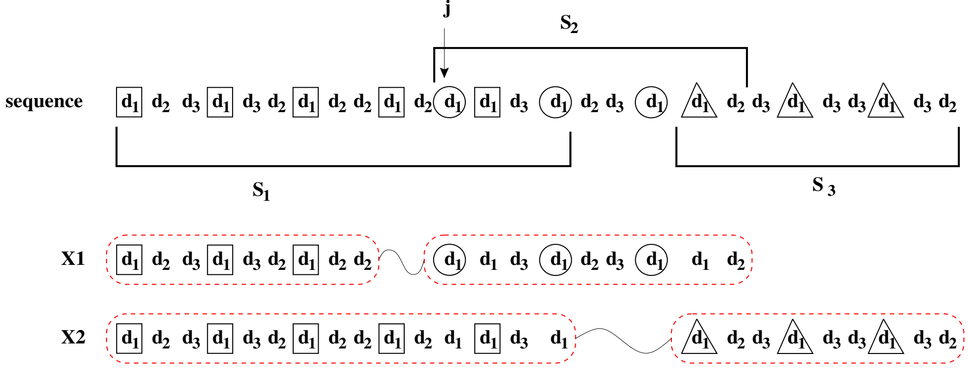


Figure 1: Example of Symbol Sequence

This indicates that we may need to track multiple ongoing subsequences simultaneously. Since the number of different assemblages (of valid segments) grows exponentially with increasing period length, the process to mine the longest subsequence becomes a daunting task (even for a very simple pattern such as $(d_1, *, *)$). To solve this problem, for a given pattern, an efficient algorithm is developed to identify subsequences that may be extended to become the longest one and organize them in such a way that the longest valid subsequence can be identified by a single scan of the input sequence and at any time only a small portion of all extendible subsequences needs to be examined.

Another innovation of our mining algorithm is that it can discover all periodic patterns regardless of the period length. Most previous research in this area focused on patterns for some pre-specified period length [12] [13] [21] or some pre-defined calendar [24]. Unfortunately, in practice, the period is not always available a priori (It is also part of what we want to mine out from the data). The stock of different merchandises may be replenished at different frequencies (which may be unknown ahead of time[1] and may also varies from time to time). A period may span over thousands of symbols in a long time series data or just a few symbols. We first introduce a distance-based pruning mechanism to discover all possible periods and the set of symbols that are likely to appear in some pattern of each possible period. In order to find the longest valid subsequence for all possible patterns, we employ a level-wise approach. The Apriori property also holds on patterns of the same period. That is, a valid segment of a pattern is also a valid segment of any pattern with fewer symbols specified in the pattern. For example, a valid segment for $(d_1, d_2, *)$ will also be one for $(d_1, *, *)$. Then, for each likely period, all valid patterns with their longest supporting subsequences can be mined via an iterative process.

In summary, we claim the following contributions in this paper.

- A pattern can be partially specified, e.g., $(d, *, *)$.
- A more flexible model of asynchronous periodic patterns is proposed to allow mining of all patterns
  - whose periods can cover a wide range and are not known a priori;
  - that are present within only a subsequence;

[1] The replenishment order of a merchandise may not be prescheduled but rather be filed whenever the inventory is low.

  – whose occurrences may be misaligned due to the insertion of some random disturbance.

- A two phase algorithm is devised to first generate potential periods by distance-based pruning followed by an iterative procedure to derive and validate candidate patterns and locate the longest valid subsequence containing each pattern.

- A segment-based approach is devised to discover the longest valid subsequence for a given pattern via a single scan of the input sequence.

- We also analyze the time and space complexity and prove the correctness of the proposed algorithm.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of recent related research to our problem. The general model is presented in Section 3. Section 4 outlines the three major steps of our algorithm. The algorithms of distance-based pruning, the singular pattern verification, and the complex pattern verification are elaborated in Sections 5, 6, 7, respectively. We discuss some limitations and extensions of our algorithm in Section 8. Section 9 presents experimental results. The conclusion is drawn in Section 10.

2 Related Work

Discovering sequential patterns was first introduced in [2] and [25]. The input data is a set of sequences, called data-sequences. Each data-sequence is a list of transactions. Typically there is a transaction time associated with each transaction. A sequential pattern also consists of transactions, i.e., sets of items. The problem is to find all sequential patterns with a user-specified minimum support, where the support of a sequential pattern is the percentage of data-sequences that contain the pattern. The surprising sequential pattern discovery is proposed in [28]. In this work, the authors search for the patterns whose occurrence is significantly greater than the expectation. The information gain is used to measure the degree of surprise (or significance) of a pattern.

Full cyclic pattern was studied in [21]. The input data to [21] is a set of transactions, each of which consists of a set of items. In addition, each transaction is tagged with an execution time. The goal is to find association rules that repeat itself throughout the input data. Han et. al. [13] presented algorithms for efficiently mining partial periodic patterns by exploring some interesting properties related to partial periodicity such as the Apriori Property and the max-subpattern hit set property. However, the proposed solution requires that the predefined period and the synchronous pattern.

In [7], Bettini et. al. proposed an algorithm to discover temporal patterns in time sequences. The basic components of the algorithm includes timed automata with granularities (TAGs) and a number of heuristics. The TAGs are for testing whether a specific temporal pattern, called a candidate complex symbol type, appears frequently in a time sequence. In addition, heuristics are used to reduce the number of candidate symbol types. These heuristics exploit the information provided by explicit and implicit temporal constraints with granularity in the given symbol structure.

The inclusion of a user defined calendar is studied in [24]. A user explicitly defines a calendar and interesting patterns are discovered based on the calendar. For example, if a user defines temporal subsequence to start on the days when the US government announces the unemployment rate as this calendar and the calendar is applied to the stock prices in New York Stock Exchange, then some interesting patterns can be discovered relating the reaction of stock prices to these announcements.

3 General Model

In this section, we formally define the model that we are investigating in this paper. Let $\Im = \{d_1, d_2, \ldots, \}$ be a set of literals and D be a sequence of literals in $\Im$. We first introduce some notations that would facilitate the discussion in the remainder of the paper.

Definition 3.1 *A* **pattern** *with* **period** l *is a sequence of* l *symbols* $(p_1, p_2, \ldots, p_l)$*, where* $p_1 \in \Im$ *and the others are either a symbol in* $\Im$ *or *, i.e.,* $p_j \in \Im \cup *$ $(2 \leq j \leq l)$*.*

Since a pattern can start anywhere in a sequence, we only need to consider patterns that start with a non "*" symbol. Here, $*$ is introduced to allow partial periodicy. In particular, we use $*$ to denote the "don't care" position(s) in a pattern [13]. A pattern P is called a i**-pattern** if exactly i positions in P are symbols from $\Im$. (The rest of the positions are filled by *.) For example, $(d_1, d_2, *)$ is a 2-pattern of period 3.

Definition 3.2 *For two patterns $P = (p_1, p_2, \ldots, p_l)$ and $P' = (p'_1, p'_2, \ldots, p'_l)$ with the same period l, P' is a* **specialization** *of P (i.e., P is a* **generalization** *of P') iff, for each position $j (1 \leq j \leq l)$, either $p_j = p'_j$ or $p_j = *$ is true.*

For example, pattern $(d_1, d_2, *)$ is considered as a specialization of $(d_1, *, *)$ and a generalization of (d_1, d_2, d_3).

Definition 3.3 *Given a pattern $P = (p_1, p_2, \ldots, p_l)$ with period l and a sequence of l literals $D' = d_1, d_2, \ldots, d_l$, we say that P* **matches** *D' (or D'* **supports** *P) iff, for each position $j (1 \leq j \leq l)$, either $p_j = *$ or $p_j = d_j$ is true. D' is also called a* **match** *of P.*

In general, given a sequence of symbols and a pattern P, multiple matches of P may exist. In Figure 2(a), D_1, D_2, ..., D_7 are seven matches of $(d_1, *, d_2)$. We say that two matches of the same period are **overlapped** iff they share some common subsequence, and are **disjoint** otherwise. For instance, D_1 and D_3 are disjoint while D_1 and D_2 are overlapped and their common subsequence is indicated by the shaded area in Figure 2(a).

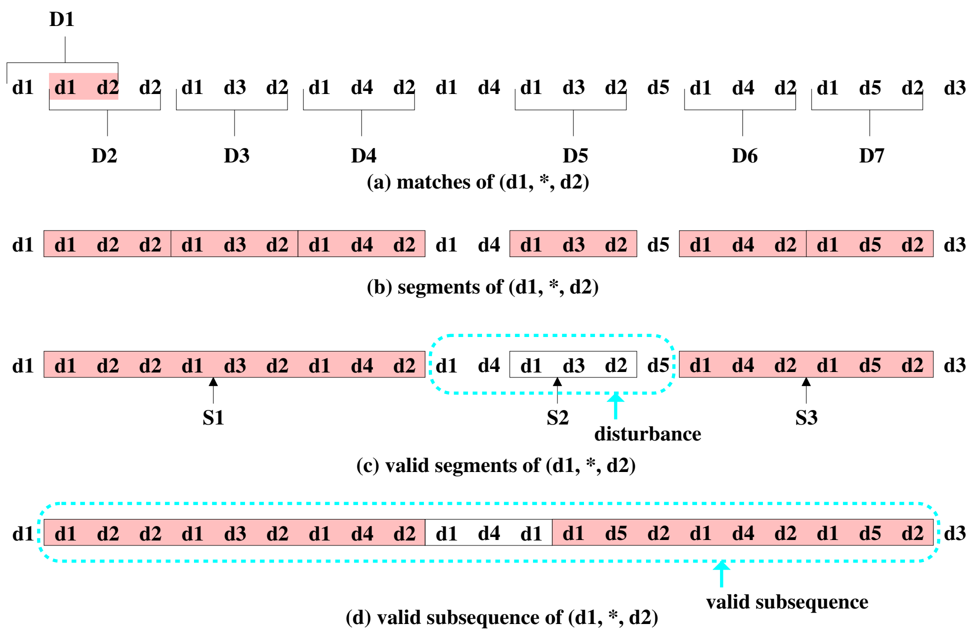


Figure 2: Example of matches and segments of $(d_1, *, d_2)$

Definition 3.4 *Given a pattern P with period l and a sequence of symbols D, a list of k $(k > 0)$ disjoint* *matches of P in D is called a* **segment** *with respect to P iff they form a contiguous subsequence of D. k is referred to as the* **number of repetitions** *of this segment.*

Segments D_2, D_3, and D_4 form a contiguous subsequence as shown in Figure 2(b). Therefore, the subsequence $d_1, d_2, d_2, d_1, d_3, d_2, d_1, d_4, d_2$ is a segment with respect to the pattern $(d_1, *, d_2)$. Note that, by definition, each match of a pattern P itself is also a segment with respect to P.

Definition 3.5 *A segment S with respect to a pattern P is a* **valid segment** *iff the number of repetitions of S (with respect to P) is at least the required minimum repetitions (i.e., min_rep).*

If the value of min_rep is set to 2, then both segments S_1 and S_3 qualify as valid segments as illustrated by shaded area in Figure 2(c). S_2 is not a valid segment since it only contains one match of $(d_1, *, d_2)$. In general, given a pattern P, any sequence of symbol can be viewed as a list of disjoint valid segments (with respect to P) interleaved by disturbance. For example, the subsequence enclosed in the dashed contour in Figure 2(c) is treated as disturbance between two valid segments S_1 and S_3.

Definition 3.6 *Given a sequence D and a pattern P, a* **valid subsequence** *in D is a set of non-overlap valid segments where the distance between any two successive valid segments does not exceed the parameter max_dis. The* **overall number of repetitions** *of a valid subsequence is equal to the sum of the repetitions of its valid segments. A valid subsequence with the most overall repetitions of P is called its* **longest valid subsequence**.

Definition 3.7 *For a sequence of symbols D, if there exists some valid subsequence with respect to a pattern, then this pattern is called a* **valid pattern**.

It follows from the definition that any valid segment itself is also a valid subsequence. If we set $max\_dis = 4$, even though S_1 and S_3 in Figure 2(c) are individually valid subsequences, there does not exist a valid subsequence containing both of them due to the violation of the maximum allowed disturbance between them. In contrast, the subsequence enclosed by dashed line in Figure 2(d) is a valid subsequence whose overall number of repetitions is 6.

For a given sequence of literals D, the parameters min_rep and max_dis, and the maximum period length L_{max}, we want to find the valid subsequence that has the most overall repetitions for each valid pattern whose period length does not exceed L_{max}.

4 Algorithm Overview

In this section, we outline strategies to tackle the problem of mining subsequences with most overall repetitions for all possible patterns.

1. *Distance-based Pruning.* For each symbol d, we examine the distance between any two occurrences of d. Let $DC_{d,l}$ be the number of times when such distance is exactly l. For each period l, the set of symbols whose $DC_{d,l}$ counters are at least min_rep are retained for the next step.

2. *Single Pattern Verification.* For each potential period l and each symbol d that passed the previous step, a candidate 1-pattern $(d_1, d_2, \ldots, d_l)$ is formed by assigning $d_1 = d$ and $d_2 = \ldots = d_l = *$. We validate all candidate patterns $(d, *, \ldots, *)$ via a single scan of the sequence. Note that any single pattern of format $(*, \ldots, d, *, \ldots, *)$ is essentially equivalent to the pattern $(d, *, \ldots, *)$ (of the same period) with a shifted starting position in the sequence. A segment-based approach is developed so that a linear scan of the input sequence is sufficient to locate the longest valid subsequence for a given pattern.

3. *Complex Pattern Verification.* An iterative process is carried out where at the ith iteration, the candidate i-patterns are first generated from the set of valid $(i-1)$-patterns, and are then validated via a scan of the data sequence.

We now elaborate each step in following sections.

5 Distance-based Pruning of Candidate Patterns

Since there are a huge number of potential patterns, $O(|\Im|^{L_{max}})$, a pruning method is needed to reduce the number of candidates. The pruning method is motivated by our observation that if a symbol d participates in some valid pattern of period l, there should be at least min_rep times that the distance between two occurrences of d is exactly l (in order to form a valid segment). So the proposed distance-based pruning method makes one pass over the data sequence to discover all possible periods and the set of symbols that are likely to appear in some pattern of each possible period. For each symbol d and period l, the number of times when the distance between two occurrences of d in the sequence is l is collected.

To perform the distance-based pruning, when scanning through the sequence, we need to maintain a moving window of the last L_{max} symbols scanned. For the next symbol, say d, we compare it with each of the symbol in the moving window. If a match occurs at the jth position, the count for period $(L_{max} - j + 1)$ of symbol d (denoted as $DC_{d,L_{max}-j+1}$) is incremented by 1. For example, in Figure 2(a), the third d_1 in the fifth position will contribute to both $D_{d_1,3}$ and $D_{d_1,4}$. Due to the generality of our model, for each occurrence of a symbol d, we need to track its distance to all of its previous occurrences within the moving window. Our model does not only allow partially specified patterns such as $(d, *, *)$ where d can also occur in the "*" position, but also recognizes patterns with repetition of the same symbol such as $(d, *, d)$. Hence it is not sufficient to just track the distance of a symbol to its last occurrence. Given a symbol d and a period l, if $DC_{d,l}$ is larger than or equal to the min_rep threshold, then it is possible that d might participate in some valid pattern of period l. We can use this property to reduce the candidate patterns significantly.

6 Longest Subsequence Identification for a Single Pattern

If a symbol d and period l pair has passed the distance-based pruning, then the longest subsequence identification (LSI) algorithm is used to discover the subsequence with the most repetitions of $(d, *, \ldots, *)$ with period l. Each occurrence of d in the sequence corresponds to a match of the pattern. If d occurs at position i, then the subsequence from position i to position $(i + l - 1)$ is a match of the pattern. Consider the pattern $(d_1, *, *)$ and the sequence in Figure 3(a). d_1 occurs 11 times, each of which corresponds to a match denoted by D_j $(1 \leq j \leq 11)$. Before presenting the algorithm, we first introduce the concept of *extendibility* and *subsequence dominance*.

Definition 6.1 *For any segment S with respect to some pattern P, let $D_1, D_2, \ldots, D_k$ be the list of matches*

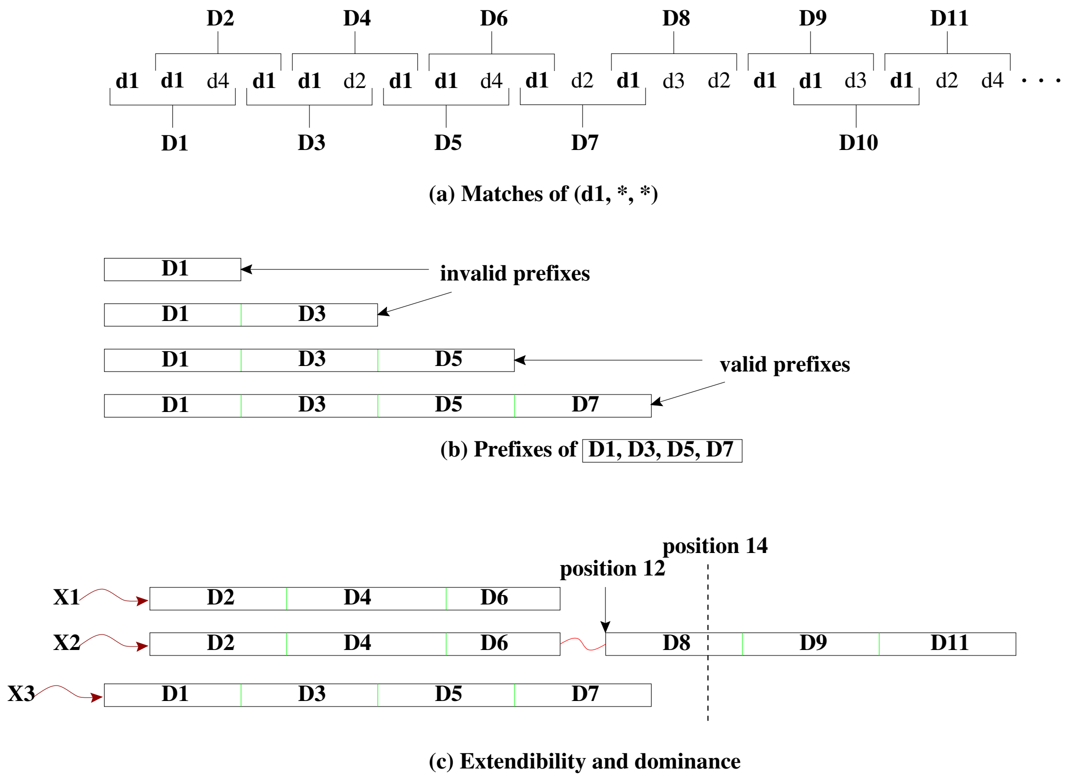


Figure 3: Example of Extendibility and Subsequence Dominance

of P that form S. Then the segment S' formed by $D_1, \ldots, D_{k'}$ is called a **prefix** *of S, where $1 \leq k' \leq k$. S is also referred to as an* **extension** *of S'. S' is called a* **valid prefix** *of S if S' is a valid segment.*

Definition 6.2 *A segment S is* **extendible** *iff it is a prefix of some other segment S' $(S' \neq S)$.*

Any segment is also a prefix of itself. For any segment S whose number of repetition is x, there exist x different prefixes of S. Figure 3(b) shows all four possible prefixes of the segment $[D_1, D_3, D_5, D_7]$ in Figure 3(a), among which the first two are not valid prefixes while the other two are valid if $min\_rep = 3$. We also say that the first three prefixes in Figure 3(b) are extendible.

Definition 6.3 *For any two valid subsequences X and Y with the same starting position, X is a* **prefix** *of Y (and Y is an* **extension** *of X) iff each valid segment in X is also a valid segment in Y, except that the last valid segment S in X may be a prefix of a valid segment S' in Y. Let j be the starting position of the first match of the pattern in Y but not in X. Then we say that X is* **extended on position j to generate** *Y.*

Definition 6.4 *Given a pattern P, a valid subsequence X is* **extendible** *if there exists another valid subsequence Y $(Y \neq X)$ such that Y is an extension of X.*

In Figure 3(c), X_1, X_2, X_3 are three valid subsequences of $(d_1, *, *)$ if $min\_rep = 3$ and $max\_dis = 5$. X_1 is a prefix of X_2 and is extended at position 12 to generate X_2. As a result, X_1 is also said to be extendible.

Definition 6.5 *Given a position i, for any two valid subsequences X and Y that end between positions $(i - max\_dis - 1)$ and i, we say that X* **dominates** *Y at position i iff the overall number of repetitions of X is greater than or equal to that of Y.*

It is clear that, at position 14 in Figure 3(c), X_3 dominates X_1. This subsequence dominance defines a total ordering among the set of valid subsequences that are considered to be *extendible* at any given position. If X dominates Y at some position i (Figure 4(a)), then the subsequence X' generated by appending Z to X starting at position i can not be shorter than the extension Y' of Y generated by appending Z to Y (Figure 4(b)). Therefore, we do not need to extend Y at position i. Note that we still have to keep Y since we may need to extend Y (to include V in Figure 4(c)) at some later position $j (j > i)$. This scenario may happen when X ends at an earlier position than Y does and X is known to be not extendible at position j. This provides the motivation and justification of the pruning technique employed in our algorithm.

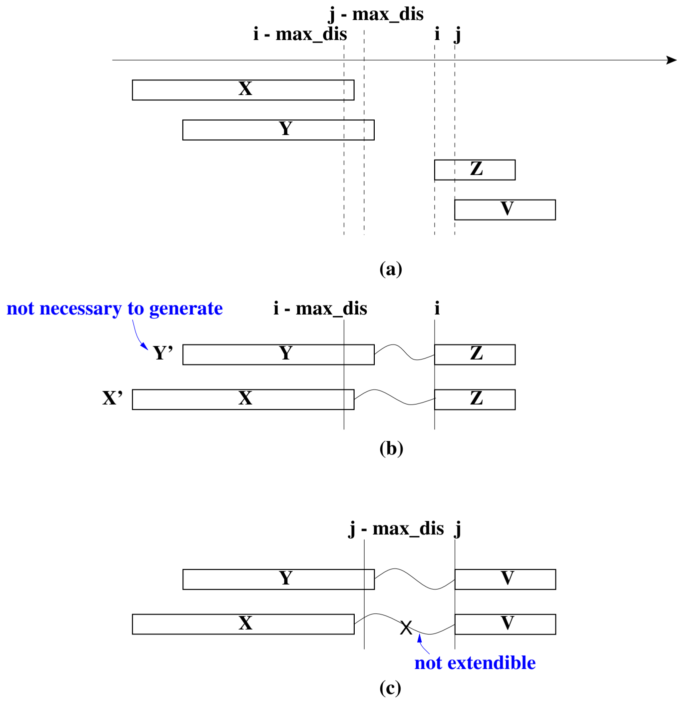


Figure 4: Subsequence Dominance

6.1 Algorithm Description

When scanning through a sequence to determine the longest subsequence containing a pattern $(d, *, \ldots, *)$, the discovery process may experience the following phases repeatedly:

- Phase A: *Segment validation*. At least one instance of $(d, *, \ldots, *)$ is found (in the latest segment of a subsequence), but the number of repetitions of the pattern is still less than min_rep.
- Phase B: *Valid segment growth*. The segment is now considered to be valid and the repetition count may continue to grow.
- Phase C: *Extension* (or *disturbance*). A valid segment may have ended. It is now going through some disturbance or noise region to see whether it can get extended to another segment of the pattern within max_dis, referred to as the *extension window*. If so, it returns to Phase A. Otherwise the subsequence terminates.

There are several challenges here. First of all, the transition point from Phase B to Phase C is not obvious. Although missing the next consecutive match clearly signals the transition to Phase C, the presence of the next match does not necessarily mean the continuation of Phase B. This is illustrated by X_1 in Figure 1 at the $(j-2)$th position. Secondly, the transition point from Phase C to Phase A is not straightforward, either. In fact, any (not just the first) d occurring within the extension window can potentially be a candidate leading to a new extension of the subsequence. For X_2 in Figure 1, it is the second d_1 in the extension window that leads to a valid segment. We thus need to develop an efficient tracking mechanism for managing the phase transitions.

Furthermore, there is also the pruning issue. There can be many overlapping subsequences of a pattern. An efficient pruning criterion needs to be developed to prune the subsequences that cannot become the longest valid subsequence. This will reduce the number of concurrent subsequences that need to be tracked. The problem here is that the longest subsequence at a particular instant may be overtaken by a shorter overlapping subsequence. This is clearly demonstrated in Figure 5(a) by X and Y. Assume that $min\_rep = 3$ and $max\_dis = 5$. After recognizing segment D_7 (before D_8 is encountered), X overtakes Y even though Y grows to be the longest valid subsequence later. However, we also observe that for any two valid subsequences X' and Y', if X' begins to dominate Y' at some position k in the sequence, any further extension of Y' will always be dominated by some extension of X' (Figure 5(b)). We can thus prune Y' after position k. A good point to check for dominance relationship is at a point when a segment Z first becomes valid. This is the point where X' that encompasses Z becomes valid and the two overlapping subsequences converge to a common tail segment.

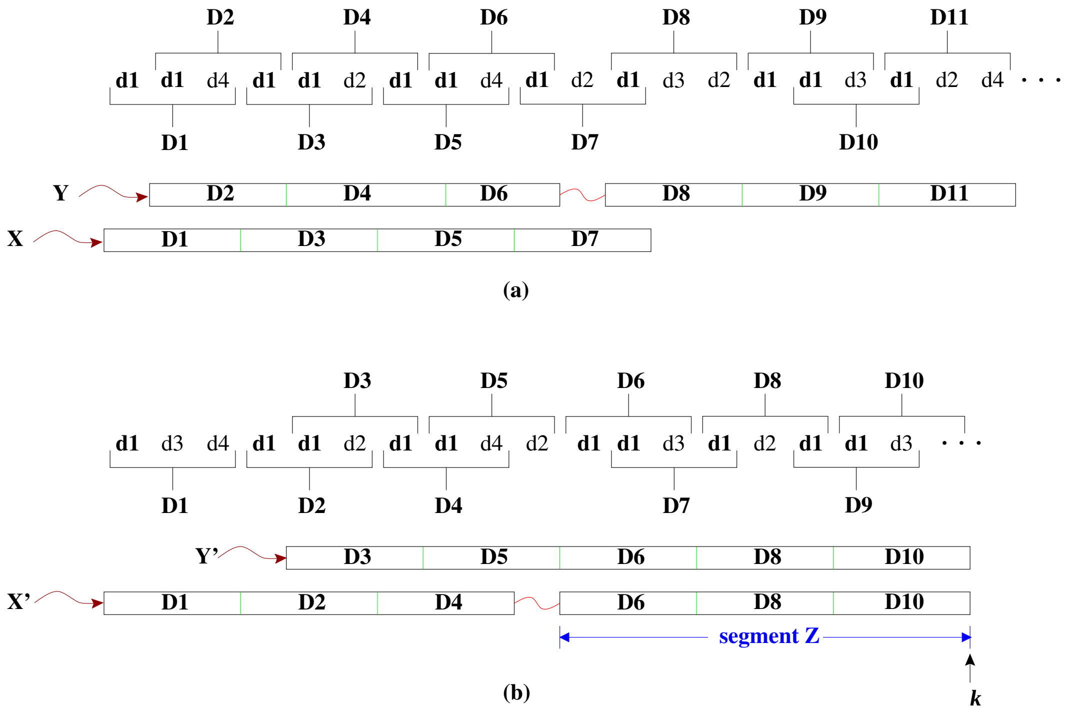


Figure 5: Valid subsequences of $(d_1, *, *)$ with $min\_rep = 3$ and $max\_dis = 5$

Inspired by above observations, the algorithm can be outlined as follows. As scanning through the input data sequence, each time a match of pattern $(d, *, \ldots, *)$ is identified (say, at position i), the set of currently *extendible* subsequences are extended according to the following principles.

- Mark the subsequences that end prior to position $i - 1$ as in Phase C.
- Only the dominating subsequence in Phase B is extended to include the newly discovered match.
- For each subsequence in Phase A, simply extend it by one repetition and check whether the transition point to Phase B is reached. If so, mark this subsequence as in Phase B. If multiple subsequences are in Phase B, then only the dominating one is retained.

- The subsequence with most repetitions in Phase B and C is identified and used to update the *longest valid subsequence* for pattern $(d, *, \ldots, *)$.

- The dominating subsequence in Phase C is also extended. The newly discovered match serves as the beginning of a new segment and the subsequence transits to Phase A.

To address the phase transition issues, the algorithm maintains three separate data structures. The $ongoing\_seq$ queue tracks all subsequences in Phases A and B. The $valid\_seq$ queue tracks the potential subsequences in Phase C. The elements in these two queues are overlapped as the transition from Phase B to Phase C is fuzzy. In fact, every subsequence in Phase B will appear in both $ongoing\_seq$ and $valid\_seq$ queues. Finally, there is the $longest\_seq$ that tracks the longest subsequence on a pattern detected so far. We now describe the contents of the various data structures after scanning the ith position of the input sequence.

- $longest\_seq$: It contains the longest valid subsequence that is known (at position i) to be not extendible. Since we have no knowledge about the data behavior after position i, only valid subsequences that end prior to position $(i - max\_dis - 1)$ are guaranteed to be not extendible at this moment. We cannot determine the extendibility of any valid subsequence that ends on or after position $(i - max\_dis - 1)$. Therefore, $longest\_seq$ is the longest valid subsequence that ends prior to position $(i - max\_dis - 1)$.

- $onging\_seq$: It contains a set of subsequences that are currently being extended, whose last segment may or may not have enough repetitions to become valid. As we will explain later, the ending position of these subsequences are between i and $(i + l - 1)$ where l is the period length[2]. Thus, we can organize them by their ending positions via a queue structure. Each *entry* in the queue holds a set of subsequences ending at the same position as illustrated in Figure 6(a). For example, if we want to verify the pattern $(d_1, *, *)$ against the sequence in Figure 6(c) with thresholds $min\_rep = 3$ and $max\_dis = 5$, the $ongoing\_seq$ queue is illustrated in Figure 6(d) after processing the d_1 that occurs in the 12th position. There are three subsequences being extended, one of which (i.e., S_1)

[2]It is obvious that there can be at most l different ending positions for these subsequences.

ends at position 14 (D_7 is the last match) while the rest (i.e., S_2 and S_3) end at position 13 (D_6 is the last match). We need to maintain both S_2 and S_3 because both of them have a chance to grow to the longest subsequence. Even though S_2 is longer than S_3, it is not a valid subsequence yet since the last segment does not meet the min_rep requirement. Therefore, we cannot discard S_3 at this moment.

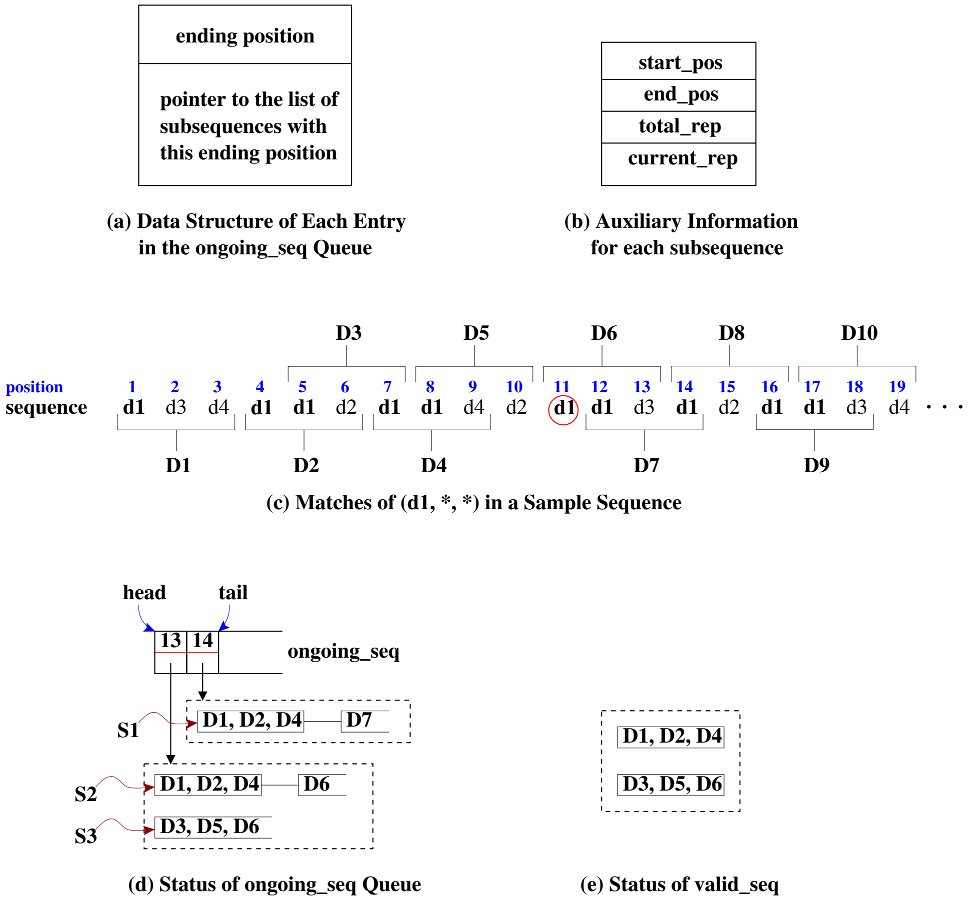


Figure 6: $ongoing\_seq$ and $valid\_seq$ data structures

- $valid\_seq$: It contains a set of valid subsequences that may be extendible. Figure 6(e) shows the $valid\_seq$ set which consists of two valid extendible subsequences. It is necessary to keep them since we may need to extend a valid subsequence multiple times by appending different matches of the pattern. For example, the segment $[D_1, D_2, D_4]$ was extended twice as shown in Figure 6(d).

For each subsequence in either $ongoing\_seq$ or $valid\_seq$, we also keep track of the starting position ($start\_pos$), the ending position (end_pos), the number of overall repetitions ($total\_rep$), and the number of repetitions of the last segment ($current\_rep$) as shown in Figure 6(b) to facilitate the tracking.

The LSI algorithm scans through the input data sequence, for each occurrence of symbol d at position i in sequence D, we have a match from position i to $(i + l - 1)$. Segments of pattern $(d, *, \ldots, *)$ are identified and can be used to extend previously generated subsequences if applicable. The following steps are taken sequentially after a match is detected at position i.

1. The $valid\_seq$ is first examined to remove all subsequences whose ending position is more than max_dis (i.e., the maximum disturbance threshold) away from the current position (i.e., position i). Note that these subsequences cannot be extended any further because of the violation of maximum disturbance requirement. Thus, it is not necessary to keep them in $valid\_seq$. At the same time, for each removed subsequence seq from $valid\_seq$, we compare it with $longest\_seq$ and update $longest\_seq$ if necessary.

2. The $ongoing\_seq$ queue is then investigated. An iteration is taken where each time the entry at the head of the queue is examined until the queue is empty or we reach an entry with ending position on or after i.

   (a) If the ending position of the subsequences in this entry is prior to $(i - 1)$, then the last segment of every subsequence in this entry cannot be extended further. For example, when the circled d_1 in Figure 6(c) is reached, we know that the segment $[D_1, D_2, D_4]$ had ended. We can simply dequeue this entry and discard it. The rationale is that we do not have to immediately extend these subsequences by initiating a new segment starting from the current match because all valid extendible subsequences in this entry are already in $valid\_seq$ and will be examined in Step 3.

   (b) If the ending position is exactly at $(i - 1)$, the last segment of the subsequences in this entry can be extended to include the current match. (The circled d_1 in Figure 6(c) also informs us that the segment $[D_3, D_5]$ can be extended to include D_6.) The following steps are taken sequentially.

      i. We append the current match to each subsequence in this entry and update the auxiliary data associated with them accordingly. The ending position of these subsequences is also updated to $(i + l - 1)$.

ii. If there are multiple valid subsequences in this entry (i.e., whose $current\_rep$ satisfies the minimum repetition requirement), then only the subsequence with the largest $total\_rep$ value is retained, the rest is discarded. It is obvious that all discarded subsequences here are dominated by the retained one. Hence, the discard would not impact the correctness of the algorithm while the efficiency can be improved.

iii. After Step ii, there can be at most one valid subsequence in this entry. If there exists one valid subsequence, then, as a potential Phase C candidate transited from Phase B, this subsequence is replicated to $valid\_seq$. (Note that it still remains in $ongoing\_seq$.) This gives a valid subsequence the opportunity to be extended in multiple ways concurrently. For example, in Figure 6(d), both S_1 and S_2 are extended from the valid segment $[D_1, D_2, D_4]$. It is necessary even if the last segment of this subsequence is still extendible. Because this subsequence may dominate all subsequences in $valid\_seq$ at some later position and in turn will be extended.

iv. Finally, this entry (now ending at position $(i + l - 1)$) is moved from the head of the $ongoing\_seq$ queue to the tail of the queue.

3. In $valid\_seq$, the subsequence seq that ends prior to position i and dominates all other subsequences with ending position prior to i is identified[3]. If seq does not end at position $(i - 1)$, then it is used to create a new subsequence new_seq by extending seq to include the current match[4]. The interval between the ending position of seq and i is treated as disturbance. new_seq is, then, inserted into the entry with ending position $(i+l-1)$ in $ongoing\_seq$ queue. (If such entry does not exist, a new entry will be created and added to the tail of the $onging\_seq$ queue.) This signals the transition of the subsequence from Phase C to Phase A.

After the entire sequence is scanned, the subsequence which has the largest $total\_rep$ value in $valid\_seq \cup \{longest\_seq\}$ is returned.

[3]In other words, seq is longest subsequence among those with ending position prior to i.

[4]No new subsequence needs to be created if seq ends at position $i - 1$ because all necessary extensions of subsequences ending at position $(i - 1)$ have been taken at Step 2(b). Step 3 is essentially designed to give the valid subsequence(s) that ends prior to position $(i - 1)$ the opportunity to be extended further by appending the current match.

6.2 Example

Figure 7(a) shows a sequence of symbols which is the same as in Figure 6(c) where the status of the various data structures after processing the 7th occurrence of d_1 at position 12 are shown in Figure 6(d) and (e) (The $longest\_seq$ is still empty.). The process of the 8th, 9th, and 10th occurrences of d_1 is illustrated in Figure 8 while the change to the data structures at each step is shown in Figure 7(b), (c), and (d).

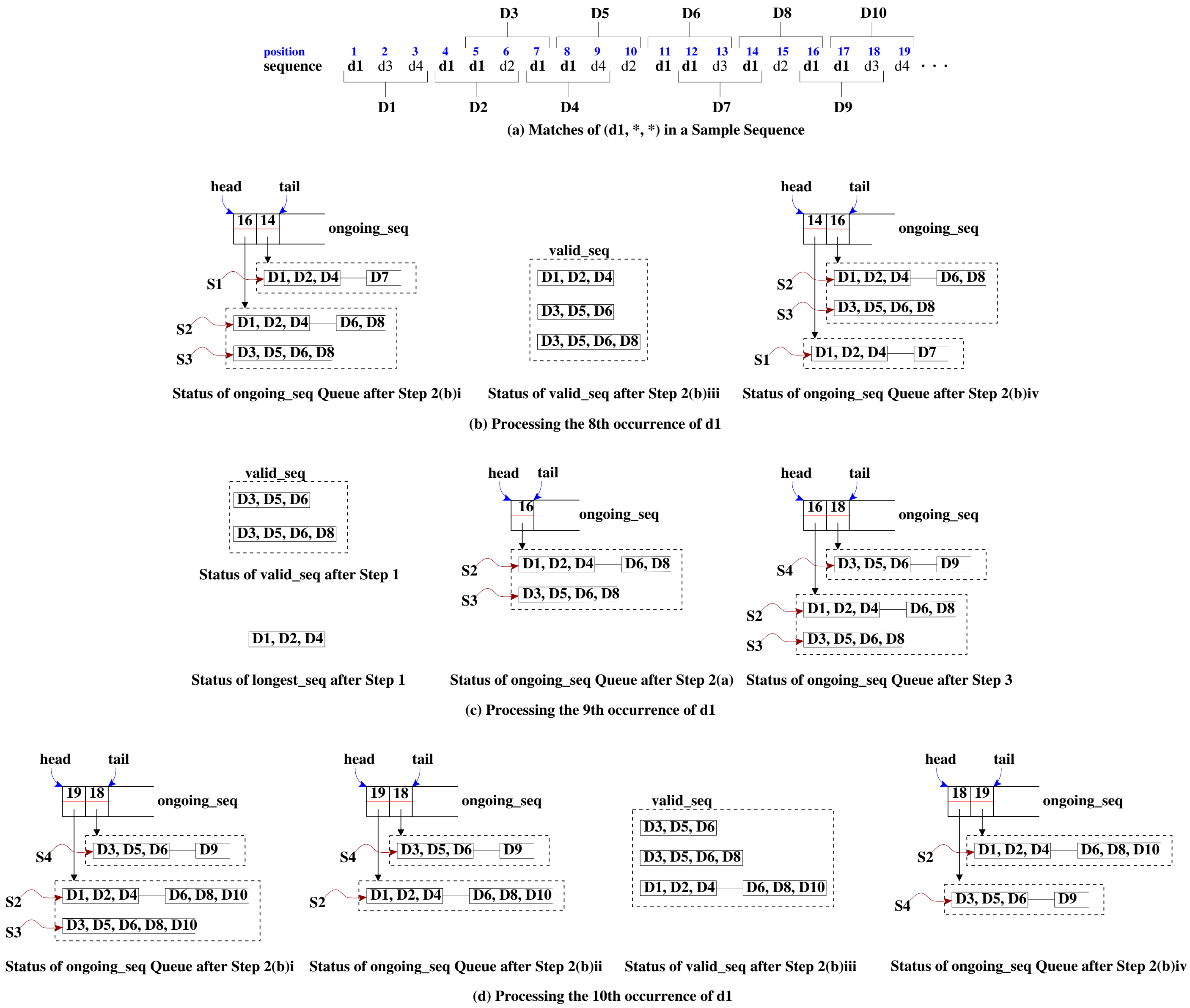


Figure 7: Status of $ongoing\_seq$, $valid\_seq$, and $longest\_seq$

From the above example, we can make the following observations.

- For $ongoing\_seq$, there can be at most l entries, each of which corresponds to a different ending

Step	8th occurrence of d1 at position 14	9th occurrence of d1 at position 16	10th occurrence of d1 at position 17
1	Both subsequences in valid_seq end less than 5 positions away from the current position (i.e., position 14). As aresult, no subsequence will be discarded from valid_seq.	D1, D2, D4 ends at position 9 which is more than 5 positions away from the current position. Thus, it is removed from valid_seq since we can not get any new extension from it. It is then put in longest_seq. (longest_seq was empty previously.)	valid_seq remains unchanged since both subsequences end less than 5 positions away from the current one.
2(a)	This step is skipped since no entry in onging_seq ends before position 13.	The entry with ending position 14 is removed from the onging_seq.	This step is skipped since no entry in onging_seq ends before position 16.
2(b)i	All subsequences in the entry with ending position 13 are extended to include D8. The ending position is also updated to 16 to reflect the change.	This step is skipped since no entry in onging_seq ends at position 15.	All subsequences in the entry with ending position 16 are extended to include D10. The ending position is also updated to 19 to reflect the change.
2(b)ii	This step is skipped since only one valid subsequence (i.e., S3) exists in this entry.		S3 is removed since S2 is also valid and longer than S3.
2(b)iii	D3, D5, D6, D8 is replicated to valid_seq.		D1, D2, D4—D6, D8, D10 is replicated to valid_seq.
2(b)iv	This entry (which ends at position 16) is moved from the head of ongoing_seq to the tail of it.		This entry (which ends at position 19) is moved from the head of ongoing_seq to the tail of it.
3	Since D3, D5, D6 ends at position 13, no new subsequence needs to be generated.	S4 is generated by extending D3, D5, D6 to include D9. A new entry is created with ending position 18 and is appended at the tail of ongoing_seq.	Since D3, D5, D6, D8 ends at position 16, no new subsequence needs to be generated.

Figure 8: Example for Illustration

position between i and $(i + l - 1)$. Furthermore, in each of these entries, each subsequence has a distinct length of the last segment (i.e., a distinct $current\_rep$). Also, there can be only one subsequence with a $current\_rep$ larger than or equal to min_rep due to Step 2(b)ii.

- For $valid\_seq$, each element has a distinct ending position between $(i - max\_dis - 1)$ and $(i + l - 1)$ due to Step 1.

6.3 Complexity Analysis

We first analyze the time complexity, then the space complexity of the LSI algorithm.

6.3.1 Time Complexity

The sizes of $valid\_seq$ and $ongoing\_seq$ are essential for analyzing the time complexity of the LSI algorithm. After processing one target symbol d at position i, at most one valid subsequence will be inserted to $valid\_seq$ (in Step 2(b)iii with ending position $(i + l - 1)$ as indicated in Step 2(b)i). It follows that every subsequence in $valid\_seq$ has different ending positions. In addition, after processing each match (starting at position i), the ending position of all subsequences in $valid\_seq$ is between $(i - max\_dis - 1)$ and $(i + l - 1)$ (inclusively). As a result, there are at most $max\_dis + l + 1$ subsequences in $valid\_seq$. Thus, the complexity of Step 1 is $O(max\_dis + l)$. Since it is invoked once for each match of the pattern, the overall time complexity of this step for processing LSI for a given symbol d is $O\left(n_d \times (max\_dis + l)\right)$ where n_d is the number of occurrences of d.

Within the entire procedure, each entry removed from the head of $ongoing\_seq$ in Step 2(a) is associated with a distinct ending position of the target pattern. Since there are at most n_d distinct ending positions (each of which corresponds to a match) in Step 2(b)i, at most n_d entries are ever removed from $ongoing\_seq$ for a given symbol d and a given period. Step 2(a) can be invoked at most n_d times. Therefore, the overall complexity for Step 2(a) is $O(n_d)$.

Consider the course of an entry r in $ongoing\_seq$ from the time it is first initialized in Step 3 to the time it is permanently discarded in Step 2(a). It is easy to show by induction that each subsequence in r has a distinct value of $current\_rep$. This claim holds trivially when r is initialized in Step 3 where only one subsequence is in r. At each subsequent time a new subsequence new_seq is added to r (in Step 3) as a result of processing a match M, the value of $current\_rep$ of new_seq is always 1 since M is the only match in the last segment of new_seq (e.g., S_2 in Figure 6(d)). In contrast, the $current\_rep$ of other subsequences in r are at least 2 after their last segments were extended to include M (e.g., S_3 in Figure 6(d)). Therefore, new_seq has a different value of $current\_rep$ from other subsequences in r. Thus, we can conclude that each subsequence in r holds a distinct value of $current\_rep$. Since there is at most one subsequence whose last segment has at least min_rep repetitions (due to Step 2(b)ii), the number of subsequences in r is bounded by min_rep. The complexity of each invocation of Step 2(b) is $O(min\_rep)$. At any time, each entry in $ongoing\_seq$ is associated with a distinct ending position. When processing a match starting at position i, at most one entry has ending position $i - 1$. The overall complexity of Step 2(b) for a given symbol d and a given period l is $O(n_d \times min\_rep)$.

As we explained before, at most $(max\_dis + l + 1)$ subsequences are in $valid\_seq$ at any time. In turn, it takes $O(max\_dis + l)$ time complexity each time Step 3 is invoked. This brings to a total complexity of $O\left(n_d \times (max\_dis + l)\right)$. In summary, the overall complexity of the LSI algorithm for a given symbol and period length l is

$$O\left(n_d \times (min\_rep + max\_dis + l)\right).$$

For a given period length l, the complexity to find the "longest" subsequence for all symbols is hence

$$O\left(\sum_{\forall d} n_d \times (min\_rep + max\_dis + l)\right)$$

which is $O\left(N \times (min\_rep + max\_dis + l)\right)$ where N is the length of the input sequence. Thus, the time complexity to discover the "longest" single-symbol subsequence for all periods and symbols is

$$O\left(N \times L_{max} \times (min\_rep + max\_dis + L_{max})\right)$$

where L_{max} is the maximum period length. This is the worst case complexity. Since the distance-based pruning may prune a large number of symbol and period pairs, the real running time could be much faster. In addition, we will propose several techniques that can reduce the time complexity of the LSI algorithm in Section 6.4.

6.3.2 Space Complexity

There are two main data structures in LSI, $ongoing\_seq$ and $valid\_seq$. For each symbol d and a given period length l, the size of $valid\_seq$ is $O(max\_dis + l)$ whereas the size of $ongoing\_seq$ is bounded by n_d since for each occurrence of d, one new subsequence is inserted to $ongoing\_seq$. Furthermore, since each entry of $ongoing\_seq$ has at most min_rep subsequences, the size of $ongoing\_seq$ is $O(min(n_d, min\_rep \times l))$. Therefore, the space complexity to find the "longest" subsequences for all symbols and a given period length l is

$$O\left((max\_dis + l) \times Num\_Symbols + min(N, min\_rep \times l \times Num\_Symbols)\right)$$

where $Num\_Symbols$ is the number of symbols in the input sequence. The overall space complexity for all possible period lengths is

$$O\left((max\_dis + L_{max}) \times Num\_Symbols \times L_{max} + min(N \times L_{max}, min\_rep \times L_{max}^2 \times Num\_Symbols)\right).$$

The above space complexity analysis is again a theoretical bound in the worst case; however, the space requirement is much smaller in practice shown in Section 8.2. Thus, in reality, all data structures can be easily fit into main memory.

6.4 Improvement of the Longest Subsequence Identification Algorithm

The time complexity of the LSI algorithm can be improved further. One way is to use a queue to store $valid\_seq$ and a heap to index all subsequences in $valid\_seq$ according to their $total\_rep$. Each time a subsequence is inserted into $valid\_seq$, it is added to the end of the queue. This would naturally make all subsequences lie in the queue in ascending order of their ending positions. Thus, Step 1 can be easily accomplished by dequeue obsolete subsequence(s) from the head of the queue. Of course, each of such operation would incur $O(\log(max\_dis + l))$ overhead to maintain the indexing heap. However, in virtue of the heap indexing, each invocation of Step 3 only requires $O(\log(max\_dis + l))$ time complexity for period length l. Therefore, the overall complexity of LSI algorithm for all period lengths and symbols is reduced to

$$O\left(N \times L_{max} \times (min\_rep + \log(max\_dis + L_{max}))\right). \tag{1}$$

6.5 Proof of Correctness

Lemma 6.1 *The last segment of any invalid subsequence removed from $ongoing\_seq$ is not extendible.*

Proof. The only place we may remove an invalid subsequence from $ongoing\_seq$ is in Step 2(a). Assume that the subsequence ends at position k ($k < i-1$). It must be true that no match starts on position $k+1$[5]. Thus, the last segment of the invalid subsequence is not extendible. □

Lemma 6.2 *At least one prefix of each longest valid subsequence has been put in both $ongoing\_seq$ and $valid\_seq$.*

Proof. Consider the starting position, say j, of a longest valid subsequence X. All valid segments starting before position $(j - min\_rep \times l)$ have to end before position $(j - max\_dis - 1)$. (Otherwise, a longer valid subsequence can be formed by extending X "backwards" to include additional valid segment(s). This contradicts the assumption that X is the longest valid subsequence.) As a result, $valid$ is empty at position j. Then a new subsequence (denoted by Y) starting at position j consisting of one match of the

[5]Otherwise, all subsequences in that entry would be extended to end at position $k + l$.

pattern is added to $ongoing\_seq$. In addition, j is the starting position of a valid segment (because it is the starting position of X). By Lemma 6.1, Y will stay in $ongoing\_seq$ until it grows to become a valid subsequence (i.e., cumulates at least min_rep repetitions). When Y is extended to a valid subsequence (denoted by Z), Z will be replicated to $valid\_seq$ because Z is the longest one in $ongoing\_seq$. (All other subsequences in $ongoing\_seq$ start later than Z.) Thus, this lemma holds. □

Lemma 6.3 *After processing each match, all valid subsequences in $ongoing\_seq$ is also in $valid\_seq$.*

Proof. All valid subsequences in $ongoing\_seq$ are generated in Step 2(b)i (some of which might be removed immediately in Step 2(b)ii). The remaining one (if applicable) is then replicated in $valid\_seq$ (Step 2(b)iii). □

By Lemmas 6.1 and 6.2, for any longest valid subsequence, either it is fully generated and used to update $longest\_seq$ or one of its valid prefix is removed from $valid\_seq$ without being extended further. Now, consider the processing of a match M starting on position i.

Lemma 6.4 *After processing a match M that starts at position i, one of the longest valid subsequences that end between position $(i - max\_dis - 1)$ and $(i + l - 1)$ is in $valid\_seq$.*

Proof. When processing a match M at positions i to $(i + l - 1)$, a valid subsequence X with ending position k $(i - max\_dis - 1 \leq k \leq i - 1)$ may not be extended to include M due to one of the following two reasons.

1. $k = i - 1$. X was removed from $ongoing\_seq$ in Step 2(b)ii because of the existence of another valid subsequence Y ending at position k in $ongoing\_seq$ such that Y dominates X. Y is chosen to be extended to include M and to be retained in $ongoing\_seq$ for potential further extension.

2. Otherwise, X is in $valid\_seq$ and is dominated by some other valid subsequence Y in $valid\_seq$, which is extended to include M and added into $ongoing\_seq$ (Step 3).

In summary, the only reason to stop extending X is that X is dominated by some other valid subsequence Y that is extended to include M and resides in $ongoing\_seq$. By Lemma 6.3, all valid subsequences in $ongoing\_seq$ is in $valid\_seq$. Therefore, after processing each match, any valid subsequence ending between position $(i - max\_dis - 1)$ and $(i + l - 1)$ is either itself in $valid\_seq$ or is dominated by some other valid subsequence in $valid\_seq$. In other words, at least one of the longest valid subsequences that end between position $(i - max\_dis - 1)$ and $(i + l - 1)$ is in $valid\_seq$. □

Lemma 6.5 *After processing a match M that starts at position i, one of the longest valid subsequences that end prior to position $(i - max\_dis - 1)$ is in $longest\_seq$.*

Proof. This proof is trivial because every time a subsequence is removed from $valid\_seq$ (due to an obsolete ending position), the $longest\_seq$ is updated if necessary. □

The following theorem is a direct inference of Lemma 6.4 and 6.5.

Theorem 6.6 *After processing the entire sequence, $longest\_seq$ holds one of the longest valid subsequences.*

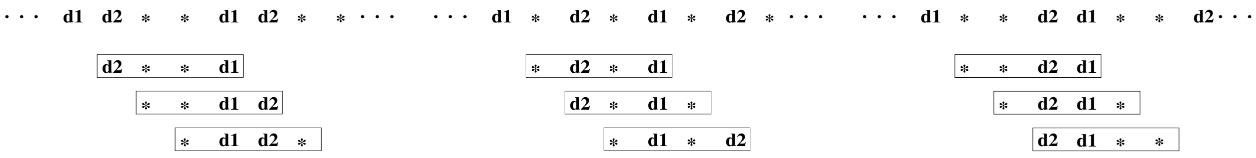
7 Complex Patterns

After discovering the single patterns, valid subsequences of different symbols may be combined to form a valid subsequence of multiple symbols of the same period. We employ a level-wise search algorithm, which generates the subsequences of i-patterns based on valid subsequences of $(i-1)$-patterns with the same period length. To efficiently prune the search space, we use two properties: one is the **symbol** property, and the other is the **segment** property.

Property 7.1 (Symbol property) *If a pattern P is valid, then all of its generalizations are valid.*

Property 7.2 (Segment property) *If $D' = d_j, d_{j+1}, d_{j+2}, \ldots, d_k$ is a valid segment for pattern P, then D' is also a valid segment of all generalizations of P.*

Since these two properties are straightforward, we would omit the proof. Based on these properties, we can prune the candidates of a valid pattern efficiently. For example, if two patterns $(d_1, *, *, *)$ and $(d_2, *, *, *)$ are valid, then three candidate 2-patterns can be generated: $(d_1, d_2, *, *)$, $(d_1, *, d_2, *)$ and $(d_1, *, *, d_2)$. As shown in Figure 9, all other 2-patterns of period 4 containing d_1 and d_2 are equivalent to



(a) equivalent patterns of (d1, d2, *, *) **(b) equivalent patterns of (d1, *, d2, *)** **(c) equivalent patterns of (d1, *, *, d2)**

Figure 9: Equivalent Complex Patterns

one of these three patterns with a shifted starting position. Similarly, $(d_1, d_2, d_3, *)$ can become a candidate 3-pattern only if $(d_1, d_2, *, *)$, $(d_1, *, d_3, *)$, and $(d_2, d_3, *, *)$ are all valid.

After the candidate set of valid i-patterns is generated, then a similar algorithm to LSI is executed to verify whether these candidates are indeed valid patterns. As a side product, the LSI algorithm also finds the valid subsequence with the most repetitions for each valid pattern.

8 Discussion

8.1 Parameters Specification

In our approach, the mining results can be effected by the choice of the two parameters min_rep and max_dis. When the parameters are not set properly, noises may be qualified as patterns. We use an iterative method to set the proper value for these two parameters. After discovering patterns for a given pair of min_rep and max_dis, we prune those discovered patterns according to the statistical significance. For example, if $(a, *, *)$ is a discovered pattern and the expected continuous repetition of this pattern in a random sequence exceeds the min_rep threshold, then we can conclude that this pattern may occur due to chance and it should be discarded. Notice that there may not be a uniform threshold of min_rep and max_dis for statistical significance since the probability of occurrence of two patterns may be difference. For instance, the occurrence probability of $(a, *, *)$ should be higher than that of $(a, b, *)$. After pruning, if the number of remaining patterns is too small, and we can adjust the parameters of min_rep and max_dis, e.g., reducing min_rep or increasing max_dis, and mine patterns again. This process terminates when there is a sufficient number of patterns discovered.

8.2 Noises

There may be many types of noises in real applications. The parameter max_dist is employed to recognize the noises between segments of perfect repetitions of a pattern. There may exist other types of noises, e.g., intra-pattern noise. Let $(a, c, *, b, *, *, b)$ be a pattern. The segment $accbaaab$ may be the occurrence of this pattern with some noise. (There is an extra symbol between the last two bs.) We can modify the definition of asynchronous pattern slightly to recognize this type of noises. If a segment is very similar to a pattern (within a certain degree), then we consider the segment as a repetition of the pattern. Without major modification, our algorithm should be able to handle this type of noise.

8.3 Extensions

In some application, e.g., sensor network, multiple events may occur simultaneously. As a result, it is possible that multiple symbols may occur in the same position within a sequence. The proposed approach can be easily extended to handle this situation. We only need to modify one step in our algorithm. When generating candidates, we need to take into account all possible subsets of symbols at a time slot. For example, if symbols A, B occurred at the same time slot, then during candidate generation phase, we need to consider four possible candidates for this time slot, $\{A\}$, $\{B\}$, $\{A, B\}$, and $\{*\}$.

The other possible extension of the asynchronous pattern is to discover possible sequential rules, e.g., "A is followed by B with a chance 50% in a given subsequence". To find this type of sequential rules, the minimum repetitions can be viewed as support. The asynchronous patterns discovered by our algorithm can be considered as the patterns that satisfy the support threshold. In the post-process step, we can verify whether the rules satisfy the confidence requirement. To quality the rule "A is followed by B with a chance 50%, then by C with a probability 75% in a given subsequence", all three patterns $(a, *, \ldots)$, $(b, *, \ldots)$, and $(c, *, \ldots)$ have to be valid for the a sufficient long portion of the sequence. Next, we can verify whether the confidence requirement (e.g., 50% and 75%) is also satisfied.

9 Experimental Results

We implemented the PatternMiner in C programming language and it is executed on an IBM RS-6000 (300 MHz CPU) with 128MB running an AIX operating system.

9.1 Real Sequence

We first apply our model to a real trace of a web access log. Scour is a web search engine specialized in multimedia content search whose URL is "http://www.scour.net". Since early of 2000, the average daily number of hits on Scour has grown over one million. A trace of all hits on Scour between March 1 and June 8 (total 100 days) were collected. The total number of accesses is over 170 million. Then the entire trace is summarized into a sequence as follows. The trace is divided into 10 minute intervals. The number of hits during each 10 minute interval is calculated. Finally, we label each interval with a symbol. For example, if the number of hits is between 0 and 4999, then this interval is labeled as a, if the number of hits is between 5000 and 9999, then this interval is labeled as b, and so on. The summarized sequence consists of 14400 occurrences of 71 distinct symbols.

There exist some interesting patterns discovered by our algorithm. When min_rep and max_dis are set to 4 and 200, respectively, there are overall 212 patterns discovered. The following is some example of discovered patterns. There exists a pattern (b,b, b) in weekdays between 3am and 8:30am EST. Another pattern (c, c, c) occurs during 11am to 5pm EST weekday.

In the above experiment, the overall length of the sequence is relatively short (14400), hence, all three mining processes are done in less than 1 minute. To further understand the behavior of our proposed asynchronous pattern mining algorithm, we constructed four long synthetic sequences and the sensitive analysis of our algorithm on these sequences is presented in the following section.

9.2 Synthetic Sequence Generation

For the purpose of evaluation of the performance of the PatternMiner, we use four synthetically generated sequences. Each sequence consists of 1024 distinct symbols and 20M occurrences of symbols. The synthetic sequence is generated as follows. First, at the beginning of the sequence, the period length l

of the next pattern is selected based on a geometric distribution with mean μ_l. The number of symbols involved in a pattern is randomly chosen between 1 and the period l. The number of valid segments is chosen according to a geometrical distribution with mean μ_s. The number of repetitions of each valid segment follows a geometrical distribution with mean μ_r. After each valid segment, the length of the disturbance is determined based on a geometrical distribution with mean μ_d. This process repeats until the length of the sequence reaches 20M. Four sequences are generated based on values of μ_l, μ_s, μ_r, and μ_d in Table 1. In the following experiments, we always set $L_{max} = 1000$ and for each period l, $max\_dis = \frac{min\_rep \times l}{4}$.

Data Set	μ_l	μ_s	μ_r	μ_d
$DS1$	5	5	50	50
$DS2$	5	5	1000	1000
$DS3$	100	1000	50	50
$DS4$	100	1000	1000	1000

Table 1: Parameters of Synthetic Data Sets

9.3 Distance-based Pruning

In this section, we are investigating the effects of distance-based pruning. Without the distance-based pruning, there could be as many as $\mid \Im \mid \times L_{max}$ different single patterns $(d, *, \ldots, *)$ $(\forall d \in \Im)$. Figure 10(a) shows the fraction of patterns eliminated by distance-based pruning. It is evident that the number of pruned patterns is a monotonically increasing function of the min_rep threshold. With a reasonable min_rep threshold, only a small portion of potential single patterns need to be validated and used to generate candidate complex pattern.

Figure 10(b) shows the space utilized by $ongoing\_seq$ and $valid\_seq$ for all patterns in the LSI algorithm. The real space utilization is far less than the theoretical bound shown in Section 6.3.2 due to the following two reasons. First a large number of candidate patterns are pruned. (More than 90% as shown in Figure 10(a).) Secondly, in the theoretical analysis, we consider the worst case for the space utilization of $ongoing\_seq$, however, in reality, the space occupied $ongoing\_seq$ is much less than the theoretical bound.

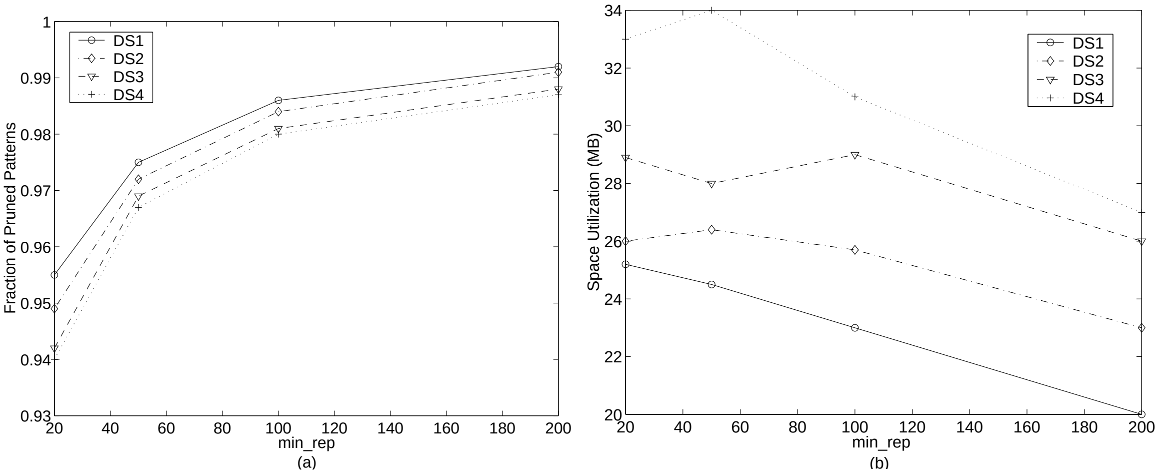


Figure 10: Effects of Distance-based Pruning

9.4 Pattern Verification

After the distance-based pruning, each remaining potential single pattern is validated through the LSI algorithm. Since the validation process (even for a given pattern) is not a trivial task, in this subsection, we demonstrate the efficiency of our LSI algorithm by comparing it with a reasonable two stage (TS) algorithm. In the TS algorithm, for a given pattern, all valid segments are first discovered, then all possible combinations of valid segments are tested and the one with the most repetition is chosen. Figure 11 shows the average elapse time of validating a pattern. (Note that the Y-axis is in log scale.) It is evident that LSI can outperform the TS algorithm by at least one order of magnitude regardless the min_rep threshold.

9.5 Overall Response Time

Figure 12 shows the overall response time of PatternMiner to find all patterns. The x-axis shows the value of min_rep whereas the y-axis shows the overall response time of PatternMiner. The higher the min_rep threshold, the shorter the overall response time. This is in contrast to Formula 1 due to the effect of distance-based pruning shown in Figure 10.

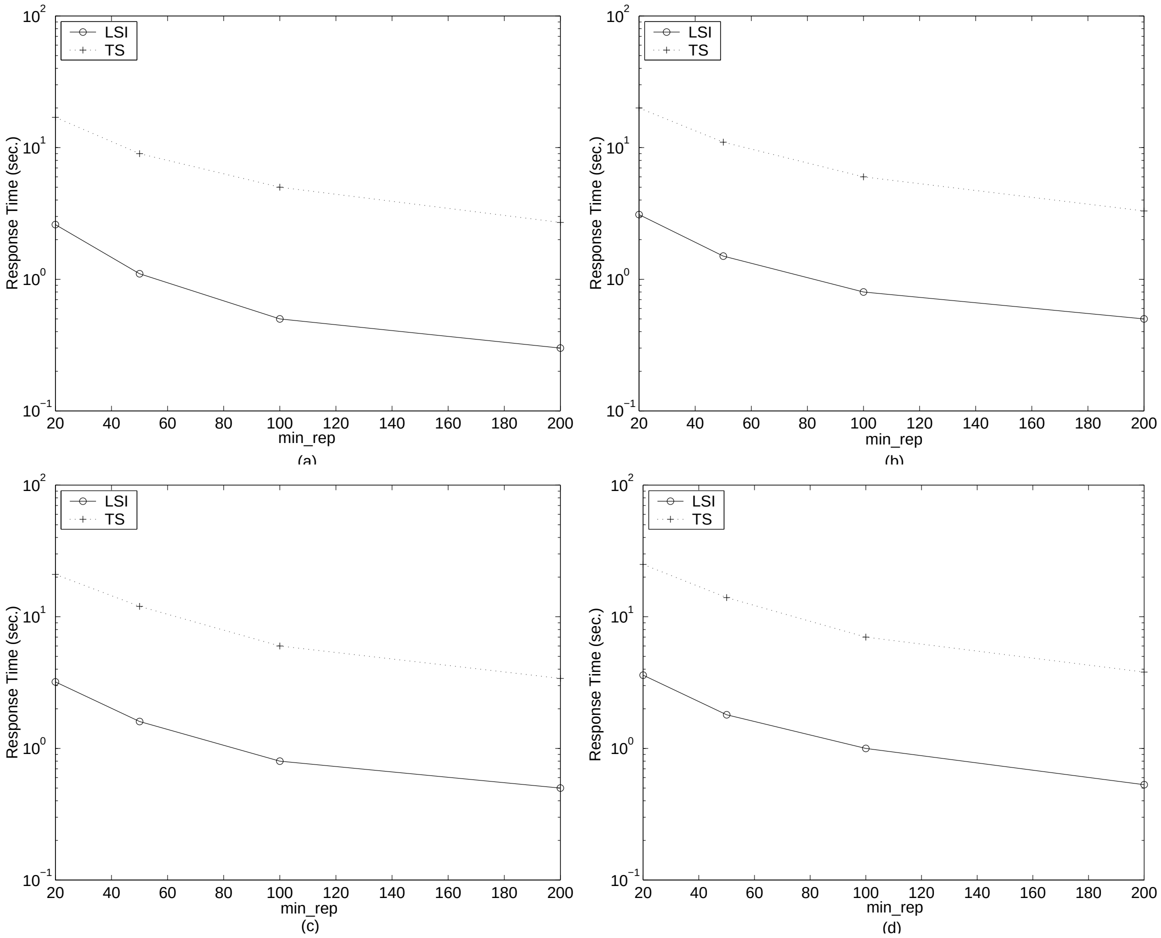


Figure 11: LSI Algorithm vs. TS Algorithm

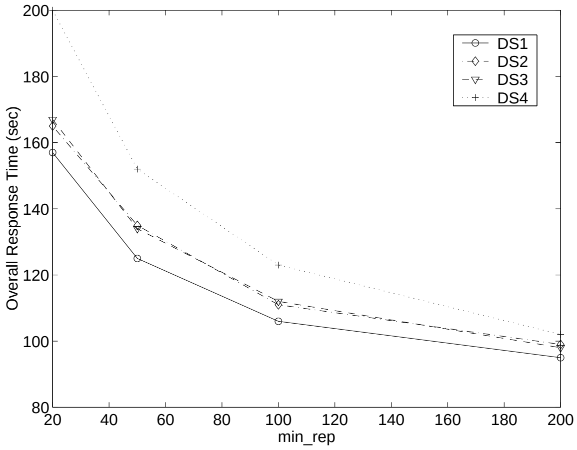


Figure 12: Overall Response Time of PatternMiner

10 Conclusion

In this paper, we propose a more flexible model of asynchronous periodic patterns to mine patterns that are of any length and may only be present within a subsequence, and whose occurrences may be shifted due to disturbance. Two parameters min_rep and max_dis are employed to specify the minimum number of repetitions required within each contiguous segment of pattern occurrences and the maximum disturbance allowed between any two successive valid segments. Upon satisfying these two requirements, the longest valid subsequence of a pattern is returned. A two phase algorithm is devised to first generate potential periods by distance-based pruning followed by an iterative procedure to derive and validate candidate patterns and locate the longest valid subsequence. We also show that this algorithm can not only provide linear time complexity with respect to the input sequence length but also achieve space efficiency. This is also demonstrated via the experimental results.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. 20th Int. Conf. on Very Large Data Bases*, 487-499, 1994.

[2] R. Agrawal and R. Srikant. Mining Sequential Patterns. *Proc. Int. Conf. on Data Engineering (ICDE)*, Taipei, Taiwan, 3-14, March 1995.

[3] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. *Proc. 21st Int. Conf. on Very large Data Bases*, 502-514, 1995.

[4] R. J. Bayardo Jr. Efficiently mining long patterns from databases. *Proc. ACM SIGMOD Conf. on Management of Data*, 85-93, 1998.

[5] J. Bentley. Programming pearls. *Communications of ACM*, vol. 27, no. 2, 865-871, 1984.

[6] D. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. *Advances in Knowledge Discovery and Data Mining*, 229-248, 1996.

[7] C. Bettini, X. S. Wang, S. Jajodia, and Jia-Ling Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transaction on Knowledge and Data Engineering*, 10(2), 222-237, 1998.

[8] K. Chan and A. W. Fu. Efficient time series matching by wavelets. *Proc. 15th Int. Conf. on Data Engineering*, 126-133, 1999.

[9] G. Das, K.-I. Lin, H. Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. *Proc. Int. Conf. on Knowledge Discovery and Datamining*, 16-22, 1998.

[10] R. Feldman, Y. Aumann, A. Amir, and H. Mannila. Efficient algorithms for discovering frequent sets in incremental databases. *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 59-66, 1997.

[11] V. Guralnik and J. Srivastava. Event detection from time series data. *Proc. ACM SIGKDD*, 33-42, 1999.

[12] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, 214-218, 1998.

[13] J. Han, G. Dong, and Y. Yin. Efficient mining partial periodic patterns in time series database. *Proc. Int. Conf. on Data Engineering*, 106-115, 1999.

[14] H. V. Jagadish, N. Koudas, and S. Muthukrishnan. Mining deviants in a time series database. *Proc. 25th Int. Conf. on Very Large Data Bases (VLDB)*, 102-113, 1999.

[15] E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. *Proc. Int. Conf. on Knowledge Discovery and Datamining*, 24-30, 1997.

[16] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting Ad Hoc queries in large datasets of time sequences. *Proc. ACM Conf. on Management of Data (SIGMOD)*, 289-300, 1997.

[17] L. Lin and T. Risch. Querying continuous time sequences. *Proc. 24th Int. Conf. on Very Large Data Base (VLDB)*, 170-181, 1998.

[18] B. Liu, W. Hsu, and Y. Ma. Mining association Rules with multiple minimum supports. *Proc. ACM SIGKDD*, 337-341, 1999.

[19] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, vol. 1, no. 3, 259-289, 1997.

[20] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. *Proc. ACM SIGKDD*, 322-326, 1999.

[21] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. *Proc. 14th Int. Conf. on Data Engineering*, 412-421, 1998.

[22] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. *Proc. 7th ACM Int. Conf. on Information and Knowledge Management*, 251-258, 1998.

[23] D. Rafiei. On similarity-based queries for time series data. *Proc. 15th Int. Conf. on Data Engineering*, 410-417, 1999.

[24] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. *Proc. 24th Intl. Conf. on Very Large Data Bases (VLDB)*, 368-379, 1998.

[25] R. Srikant and R. Agrawal. Mining sequential patterns: generalizations and performance improvements. *Proc. 5th Int. Conf. on Extending Database Technology (EDBT)*, 3-17, 1996.

[26] S. Thomas and S. Sarawagi. Mining generalized association rules and sequential patterns using SQL queries. *Prof. of 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD98)*, pp. 344-348, 1998.

[27] R. Wetprasit and A. Sattar. Temporal Reasoning with qualitative and quantitative information about points and durations. *Proc. 15th National Conf. on Artificial Intelligence (AAAI-98)*, 656-663, 1998.

[28] J. Yang, W. Wang, and P. Yu. Infominer: mining surprising periodic patterns. *Proc. of 7th Intl. Conf. on Knowledge Discovery and Data Mining (KDD01)*, pp. 395-400, 2001.