# Mining Asynchronous Periodic Patterns in Time Series Data

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

Wei Wang
IBM Watson Research Center
ww1@us.ibm.com

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

## ABSTRACT

Most previous work on periodicy detection has focused on mining synchronous periodic patterns and does 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 patterns 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.

## Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Management—*database applications*

## General Terms

Asynchronous periodic pattern, Segment-based approach

## 1. INTRODUCTION

Periodicy detection of time series data is a challenging problem of great importance in many real applications. Most previous research in this area has 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 [1]. 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 an 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 normally a month. The refill order is filed at the beginning of each month. But suppose a major outbreak of flu causes an early 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. 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 patterns 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 as large as *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. For patterns satisfying these two requirements, our model will return the subsequence with the maximum overall repetitions. Note that, due to the presence of disturbances, some subsequent valid segments may not be well synchronized with the previous ones. This in turn would impose a great challenge in the mining process.

Similar to [1], 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 no strong regularity presents for the rest of the month. However, since we also allow the shifted occurrence of a valid segment, this flexible model poses a difficult problem to be solved. For a given 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(b), with $min\_rep = 3$, $S_1$, $S_2$, $S_3$, and $S_4$ are four valid segments of the pattern $P = (d_1, *, *)$. If $max\_dis = 6$, then $X$ in Figure 1(d) is the longest subsequence before $S_4$ is considered, which in turn makes $Y$ the longest one. If we only look at the symbol sequence up to position 10 without looking ahead in the sequence, it is very difficult to determine whether we should switch to $S_2$ or continue on $S_1$. 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, a segment-based algorithm is developed to identify subsequences that may be extended to become the longest one and maintain them in such a way that the longest valid subsequence can be identified by a single scan of the input sequence.

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 [1] [2] or some pre-defined calendar [3]. 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). A period may span 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. 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.

- 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 (1) whose periods can cover a wide range and are not known a priori; (2) that present within only a subsequence; and (3) 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.

The remainder of this paper is organized as follows. The general model and the algorithm outline are presented in Section 2. The algorithms of distance-based pruning, the singular pattern verification, and the complex pattern verification are elaborated in Sections 3, 4, 5, respectively. Section 6 presents experimental results. The conclusion is drawn in Section 7.

## 2. GENERAL MODEL

Let $\Im = \{d_1, d_2, \}$ be a set of literals and $D$ be a sequence of literals in $\Im$. 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 [1]. 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.

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 $P$ (or $P$ 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 $(d_1, d_2, d_3)$. 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 1(a), $D_1$, $D_2$, $\ldots$, $D_{19}$ are 19 matches of $(d_1, *, *)$.

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. $D_1$, $D_2$, $D_4$, $D_6$, and $D_8$ form a contiguous portion as shown in Figure 1(b). Therefore, the subsequence $d_1, d_2, d_4$, $d_1, d_3, d_1$, $d_1, d_3, d_1$, $d_1, d_2, d_1$, $d_1, d_3, d_1$ is a segment with respect to the pattern $(d_1, *, *)$. By definition, each match of a pattern $P$ itself is also a segment with respect to $P$. 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 $min\_rep = 5$, then segments $S_1, S_3$ and $S_4$ qualify as valid segments in Figure 1(b). $S_2$ is not a valid segment since it only contains 4 matches of $(d_1, *, *)$. 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 disturbances.

Given a sequence $D$ and a pattern $P$, a **valid subsequence** in $D$ is a set of non-overlapping 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**. 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 $max\_dis = 6$, even though $S_1$ and $S_4$ in Figure 1(b) 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, $S_2$ and $S_4$ can be included in one valid subsequence (Figure 1(c)) whose overall number of repetitions is 10.

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 $L_{max}$. A threefold algorithm is devised.

1. *Distance-based Pruning.* For each symbol $d$, we examine the distance between any two occurrences of $d$. Let $C_{d,l}$ be the number of times when such distance is exactly $l$. For each period $l$, the set of symbols whose $C_{d,l}$ counters are at least $min\_rep$ are retained.
2. *Single Pattern Verification.* For each potential period $l$ and each symbol $d$ that passed the previous step, a candidate 1-pattern $(d_1, \ldots, d_l)$ is formed by assigning $d_1 = d$ and $d_2 = \cdots = d_l = *$. A segment-based approach is used to validate all candidate patterns $(d, *, \ldots, *)$ via a single scan of the sequence. (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.)
3. *Complex Pattern Verification.* An iterative process is carried out where at the $i$th 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.

## 3. 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.

Our model not only allows 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. 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 $j$th position, the count for period $(L_{max} - j + 1)$ of symbol $d$ (denoted as $C_{d,L_{max}-j+1}$) is incremented by 1. For example, in Figure 1(a), the third $d_1$ in the sixth position will contribute to both $C_{d_1,2}$ and $C_{d_1,5}$. 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. Given a symbol $d$ and a period $l$, if $C_{d,l}$ 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.

## 4. LONGEST SUBSEQUENCE IDENTIFICATION FOR A SINGLE PATTERN

If a symbol $d$ and period $l$ pair has passed the distance-based pruning, then the segment-based approach (SB) is used to discover the subsequence with the most repetitions of $(d, *, \ldots, *)$ with period $l$. Before presenting the algorithm we introduce the concept of *extendibility* and *subsequence dominance*. At a given position $i$, a subsequence $X$ with ending position $j$ $(j < i)$ is *extendible* if $j \geq i - max\_dis - 1$. When scanning through the symbol sequence, at any position, the set of extendible subsequences consists of those subsequences that still have potential to "grow". For example, at position 20 in Figure 1(c), both $X$ and $Y$ are extendible if $max\_dis = 6$ For any two valid subsequences $X$ and $Y$, we say that $X$ *dominates* $Y$ iff the number of repetitions of $X$ is greater than or equal to that of $Y$. Clearly, $X$ dominates $Y$ in Figure 1(c) at position 20.

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 1(c)), then the subsequence $X'$ generated by appending $Z$ to $X$ starting at position $i$ cannot be shorter than the extension $Y'$ of $Y$ generated by appending $Z$ to $Y$. 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 1(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.

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 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 $Y$ in Figure 1(d) at position 10. Secondly, the transition point from Phase C to 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 $Y$ in Figure 1(d), it is the second $d_1$ (at position 25) in the extension window that leads to a longer 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 1(d) by $X$ and $Y$. Assume that $min\_rep = 3$ and $max\_dis = 6$. At position 29, after recognizing $S_3$ as a valid segment (before $S_4$ is recognized), $X$ 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 $Y'$ that end at the same position $k$ in the sequence and $X'$ dominates $Y'$, any further extension of $Y'$ will always be dominated by some extension of $X'$ (Figure 1(e)). We can thus prune $Y'$ after position $k$. A good point to check for dominance relationship is 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.

The SB algorithm scans 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.

[4] shows that with carefully designed data structures, a single scan of the sequence is sufficient to locate the longest subsequence with approximately linear storage requirement.

## 5. 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:

- **Symbol property**. If a pattern $P$ is valid, then all of its generalizations are valid.
- **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 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 $(d_1, *, *, d_2)$. 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 the SB algorithm is executed to verify whether these candidates are indeed valid patterns. The SB algorithm also finds the valid subsequence with the most repetitions for each valid pattern.

## 6. EXPERIMENTAL RESULTS

We implement the prototype in C and execute on an IBM RS-6000 (300 MHz CPU) with 128MB running an AIX operating system. In the following experiments, we always set $L_{max} = 1000$ and for each period $l$, $max\_dis = \frac{min\_rep \times l}{4}$. For the purpose of performance evaluation, we use a synthetically generated sequence consisting of 1024 distinct symbols and 20M occurrences of symbols. The synthetic sequence is generated as follows. The period length $l$ of the next pattern is selected based on a geometric distribution with mean 50. 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 50. The number of repetitions of each valid segment follows a geometrical distribution with mean 1000. After each valid segment, the length of the disturbance is determined based on a geometrical distribution with mean 1000. This process repeats until 20M symbols are generated.

We first investigate the effects of distance-based pruning. Without the distance-based pruning, there could be as many as $|\Im| \times L_{max}$ different single patterns $(d, *, \ldots, *)$ $(\forall d \in \Im)$. Figure 2(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 patterns.

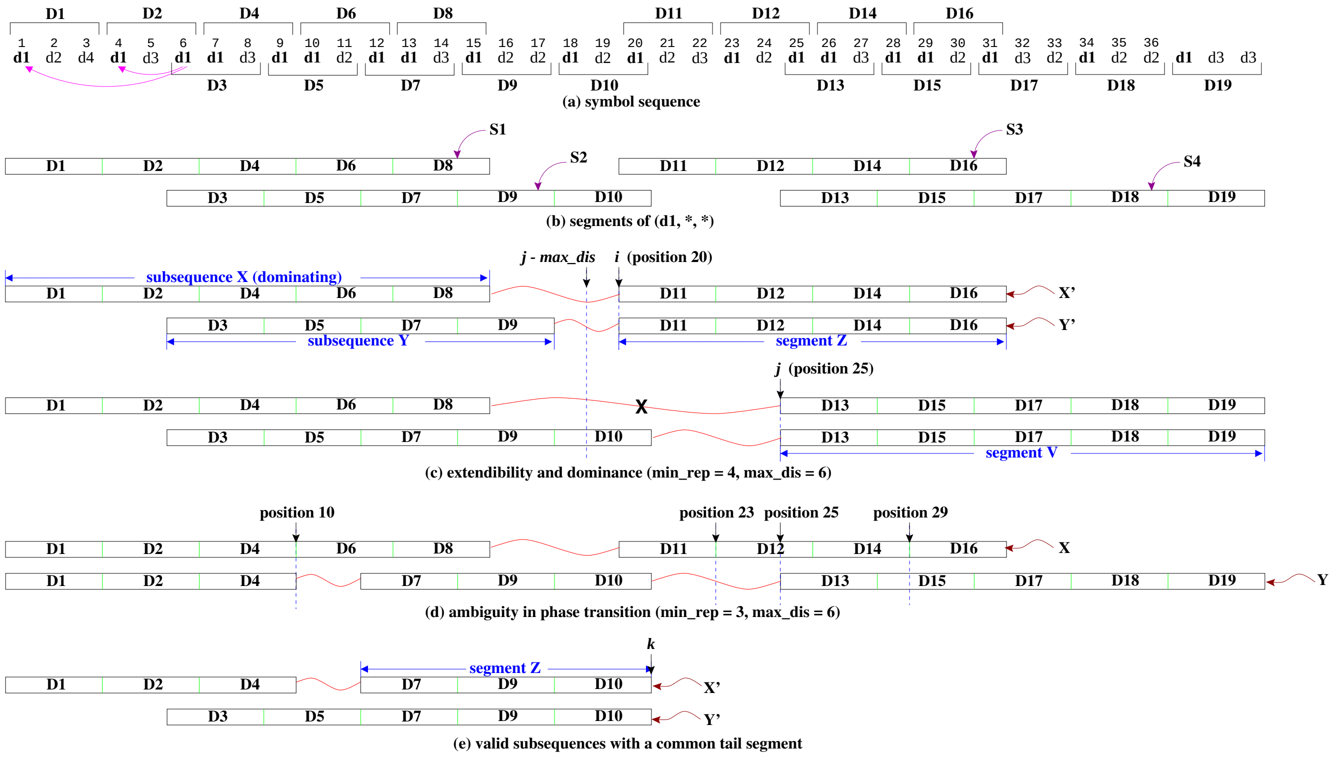


Figure 1: Valid subsequences of $(d_1, *, *)$

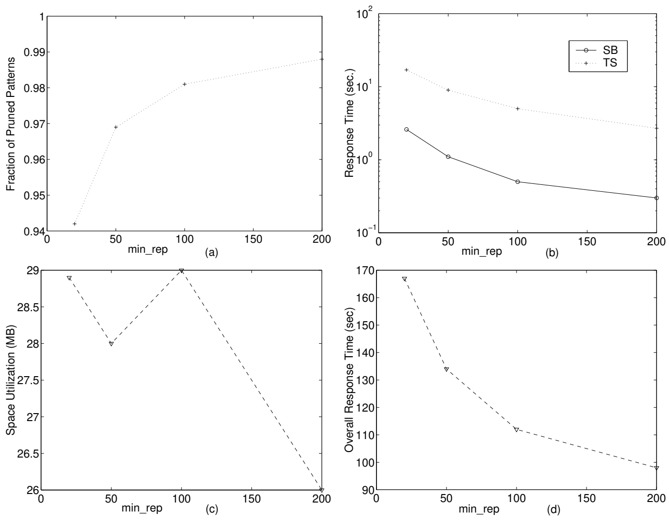


Figure 2: Effects of SB algorithm

After the distance-based pruning, each remaining potential single pattern is validated through the SB algorithm. Since the validation process (even for a given pattern) is not a trivial task, we demonstrate the efficiency of our SB 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 2(b) shows the average elapse time of validating a pattern. (Note that the Y-axis is in log scale.) It is evident that SB algorithm can outperform the TS algorithm by at least one order of magnitude regardless the *min_rep* threshold. Figure 2(c) and (d) show the space utilization in the SB algorithm and the overall response time to find all patterns, respectively. Clearly, the higher the *min_rep* threshold, the shorter the overall response time.

# 7. CONCLUSION

In this paper, we propose a 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 qualify valid patterns and the longest valid subsequence is returned for each valid pattern. 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.

# 8. REFERENCES

[1] J. Han, G. Dong, and Y. Yin. Efficient mining partial periodic patterns in time series database. *Proc. 15th ICDE*, 106-115, 1999.

[2] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. *Proc. 14th ICDE*, 412-421, 1998.

[3] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. *Proc. 24th VLDB*, 368-379, 1998.

[4] J. Yang, W. Wang, and P. S. Yu. Mining asynchronous periodic patterns in time series data. *IBM Research Report*, 2000.