

# Discovery of Generalized Local Patterns in Long Sequence

Xiaoming Jin, Yuchang Lu, Chunyi Shi

Computer Science and Technology Dept., Tsinghua University

The State Key Laboratory of Intelligent Technology and System, Beijing, China, 100084

xmjn00@mails.tsinghua.edu.cn lyc@tsinghua.edu.cn scy@est4.cs.tsinghua.edu.cn

## ABSTRACT

In recent years, there has been increased interest in using data mining techniques to extract frequent patterns from temporal sequences. Previous work mainly considers finding global patterns, where every record in the temporal sequence contributes to support the patterns. However, local patterns, which are frequent only in some time durations, are actually very common in practice and are potentially very useful. In this paper, we present a problem class that is the discovery of local sequential patterns with the format “if  $A$  occurs, then  $B$  occurs within time  $T$ ”. We describe an index structure that support efficient locating and counting of the instances of local patterns. Based on the index, we propose a two-phase method for efficiently mining of local patterns. We have analyzed the behavior of the problem and evaluated the performance of our algorithm with both synthetic and real data. The results correspond with the definition of our problem and reveal a kind of novel knowledge. In addition, the experiment verified the superiority of our approach.

## Keywords.

Generalized local sequential pattern, temporal sequence, data mining algorithm

## 1. INTRODUCTION

Temporal sequences that are lists of transaction records ordered by transaction time constitute a large part of data stored in various information systems. Well-known examples could be derived from sales records, stock prices, weather data, medical data, etc. In recent years, there has been increased interest in using data mining techniques to extract frequent patterns from temporal sequences or from the discretized versions of time series data [14], e.g. “The price of stock  $A$  goes up and falls the next day, then it goes up the third day”.

Previous work on frequent pattern discovery has mainly considered finding global patterns, where every record in the temporal sequence contributes to the pattern. However, local patterns found only in a subsequence of the entire sequence are actually very common in practice. For example, “In the sale records of a supermarket, a customer always buys biscuits followed by soda in the summer, but biscuits followed by milk in winter.” Compared with global patterns, such local patterns reveal another kind of

knowledge. There is a broad consensus that the success of data mining will depend critically on the ability to go beyond obvious patterns and find novel and useful patterns [12]. Knowing which pattern and in which time period is frequent could be equally if not more useful than simply knowing whether a pattern is frequent.

Consider the data-mining problems in stock market where the raw data is a set of long sequences collected daily over years indicting the price movements of each stock, there are many influences on the price movements, e.g. political policies, economic environment, society environment, etc. Apparently, these influences are time varying, therefore the patterns of price movements related to the influencing factors are also time varying. Thus efficient discovery of the temporal patterns and corresponding temporal features become a crucial problem. For example, “In summer, if the stock price of a game producer goes up and stays about level for two days, then it will go up in three days”. Then we observe that there may be some correlation of price behavior from July to September so that we can plan buy-sell strategy appropriately in that season, whereas we will not be confused by this pattern when we make a decision for the rest time.

To our knowledge, this problem has not been well considered in the KDD field. The problem of mining temporal association rules and mining of second order knowledge seems alike to the problem we consider, but the formats of both the database and the knowledge are essentially different. In this paper, we introduce a problem class that is the discovery of *generalized local sequential patterns* (G-LSP), with the format “if  $A$  occurs, then  $B$  occurs within time  $T$ ”, from a long sequence. The problem has a two-dimensional solution space consisting of patterns and temporal features, therefore it is impractical that use traditional methods on this problem directly. Our approach is using a suffix-tree-like index structure to support efficient locating and counting of the instances of local patterns. By using the index, we propose a two-phase method for mining G-LSPs. In addition, a “*divide and discovery*” strategy is proposed to restrict the growing of the time and storage expense. We have analyzed the behavior of the problem and evaluated the performance of our algorithm with both synthetic and real data. The results

correspond with the definition of our problem and can really reveal a kind of novel knowledge.

The rest of the paper is organized as follows: Section 2 discusses some related work. Section 3 formally defines the problem. Section 4 describes our method for this problem. Section 5 presents our experimental results. Finally, Section 6 offers some concluding remarks.

## 2. RELATED WORK

Discovering patterns in sequence data was first introduced in the artificial intelligence area [2] and received more attention in the field of data mining and knowledge discovery in databases. The problem of mining sequential patterns was proposed in [3]. In work [4], the problem was generalized, and the GSP algorithm, which scales linearly with the number of data-sequences, was proposed for discovering sequential patterns. An incremental discovery algorithm for sequential patterns was proposed in [5].

Beside the methods for discovering sequential patterns, there is some research work on mining for patterns with different format, or mining database with different structure. [8] introduced the problem of discovering temporal patterns in multiple granularities. [6][13] proposed methods for discovering frequent episodes in sequences. In [7] interval-based events were considered, where the duration of events was regarded as a temporal constraint in the discovery process. We also consider the duration of patterns in our approach, but it serves as the results of the mining process revealing the pattern distribution.

All the algorithms used in the work described above, were designed and optimized for the purpose of discovering global frequent patterns. A practicable approach for local patterns is sliding a window through the sequence, and mining for global patterns in each window. This approach has been used in some applications and the time complexity of it is not bad. However it can only find a small part of all the local patterns, of which the valid subsequences are of the same length.

A method that finds all local patterns, as we defined in section 3, can be derived by applying the previous methods to preprocessed data. However, both which pattern exists and in which periods of time a pattern exhibits frequent are unknown beforehand. The time complexity of this method is usually poor. For example, we may retrieve all the possible subsequences, and use previous algorithms to mine patterns in each subsequence. The number of possible subsequences of an  $N$  length sequence is  $(1+2+\dots+N)=(N+1)\cdot N/2$ , which means we need to run the algorithm for global patterns for  $(N+1)\cdot N/2$  times.

Other related research work includes the discovery of temporal association rules [17][18][19] and the discovery of partial periodic patterns [20]. These problems seem alike to what we consider, but both the mining goal and the formats of the knowledge are essentially different.

## 3. PROBLEM DESCRIPTION

In this paper, we consider the database that consists of a long temporal sequence. A sequence  $S=A_1, A_2, \dots, A_m$  is a list of records ordered by position number. In actual applications, records are descriptions of interesting events that happened sequentially. Without losing generality, we represent  $S$  by a  $\$$ -terminated sequence of symbols from an alphabet  $\Sigma=\{a_1, \dots, a_k\}$ , where each symbol uniquely represents a record at a time point. A subsequence,  $S[sp, ep]=A_{sp}, \dots, A_{ep}$ , is a continuous part of the original sequence. Given a time duration represented by the starting position  $sp$  and the ending position  $ep$  relative to the sequences, we use the pattern format: if  $A$  occurs, then  $B$  occurs within time  $T$  in subsequence  $S[sp, ep]$  where  $A$  and  $B$  are two subsequence, i.e.

$$s[pa, pa+|A|-1]=A \wedge s[pb, pb+|B|-1]=B \\ \wedge s[pb, pb+|B|-1] \in s[pa, pa+T-1]$$

where  $s=S[sp, ep]$  and  $|X|$  denotes the length of subsequence  $X$ , i.e. the number of symbols in  $X$ . We denote the pattern as  $A_{(T)} \rightarrow B[sp, ep]$ . The goal of our problem is to find all such patterns that happen frequently through searching in the sequence.

To evaluate the frequency of local patterns, we use measurements *local frequency*, *local support* and *local confidence*. The *local frequency* of subsequence  $A$  in subsequence  $s$  is the number of occurrences of  $A$  in  $s$ . The *local frequency* of pattern  $A_{(T)} \rightarrow B(s)$  is the number of occurrences of the pattern in  $s$ . We denote it as:

$$Lf(A, s) = |\{i \mid s[i, i+|A|-1] = A\}|$$

$$Lf(A_{(T)} \rightarrow B, s) = |\{ \langle pa, pb \rangle \mid s[pa, pa+|A|-1] = A \\ \wedge s[pb, pb+|B|-1] = B \\ \wedge s[pb, pb+|B|-1] \in s[pa, pa+T-1] \}|$$

The *local support* of subsequence  $A$  and that of the pattern  $A_{(T)} \rightarrow B$  in subsequence  $s$  are defined as follows:

$$Lsupp(A, s) = Lf(A, s) / |s|$$

$$Lsupp(A_{(T)} \rightarrow B, s) = Lf(A_{(T)} \rightarrow B, s) / |s|$$

The *local confidence* of  $A_{(T)} \rightarrow B$  in subsequence  $s$  is the ratio of the *local frequency* of  $A_{(T)} \rightarrow B$  in  $s$  over the *local frequency* of  $A$  in  $s$ . We denote it as:

$$Lconf(A_{(T)} \rightarrow B, s) = Lf(A_{(T)} \rightarrow B, s) / Lf(A, s)$$

Given a *maximal distance*  $T$ , a *minimal support*  $\mathbf{ms}$  and a *minimal confidence*  $\mathbf{mc}$ , if the *local support* of a pattern  $A_{(T)} \rightarrow B(s)$  is no less than  $\mathbf{ms}$  and the *local confidence* of that pattern is no less than  $\mathbf{mc}$ , we consider the pattern as a *generalized local sequential pattern* (G-LSP). Then the mining problem is defined as follows: Given a sequence  $S$ , find all  $\langle A, B, s \rangle$  so that  $A_{(T)} \rightarrow B(s)$  is satisfied with:

$$Lsupp(A_{(T)} \rightarrow B(s)) \geq \mathbf{ms} \text{ and } Lconf(A_{(T)} \rightarrow B(s)) \geq \mathbf{mc}$$

where  $s$  is a subsequence in  $S$ .

**Example1.**  $S="udusdusduddsdsuudsdsu\$"$ ,  $\mathbf{ms}=0.3$ ,  $\mathbf{mc}=0.5$ ,  $T=3$ , we could find  $\langle u, d, "udusdusdud" \rangle$  is a G-

LSP, because  $L_{\text{supp}}(u \rightarrow d(\text{"udusdusdud"})) = 4/10 = 0.4 \geq \text{ms}$  and  $L_{\text{conf}}(u \rightarrow d(\text{"udusdusdud"})) = 1 \geq \text{mc}$ . But  $u \rightarrow d$  is not a frequent pattern for the whole sequence, because the support of  $u \rightarrow d$  in  $S$  is  $6/23 < \text{ms}$ . We could find some other G-LSPs, e.g.  $\langle s, u, \text{"suuudsdsu"} \rangle$  with  $L_{\text{supp}}(s \rightarrow u(\text{"suuudsdsu"})) = 4/10 \geq \text{ms}$ ,  $L_{\text{conf}}(s \rightarrow u(\text{"suuudsdsu"})) = 4/4 \geq \text{mc}$ .

#### 4. METHOD FOR G-LSP DISCOVERY

Our method for discovering G-LSPs is: going through the sequence step by step and carries out a mining process in each step. Each mining process includes two phases, one is candidate generation, and the other is G-LSP generation. In the first phase, a suffix-tree-like index structure that we term the local pattern tree (LP-tree) is maintained for efficient locating of instances of patterns and the G-LSP candidates is derived. Then in the second phase, each candidate is verified, and G-LSPs are generated.

For the rest of this paper, we shall use the following notational conventions:  $\text{locus}(A)$  denotes the first node in the indexing tree encountered after  $A$  is spelled out;  $\text{subsequence}(t)$  denotes the subsequence spelled out in the suffix tree by following the path from root to node  $t$ ;  $T(A)$  denotes the sub-tree of  $T$  of which the root is  $\text{Locus}(A)$  and  $T(t)$  is the sub-tree of which the root is node  $t$ ;  $\{\text{Leaf}(t)\}$  denotes the set of all leaf nodes of  $T(t)$ .

##### 4.1 Index Structure

For the efficient discovery of local sequential patterns, we propose using an index structure that we term local pattern tree (LP tree), which consists of a standard suffix tree and a leaf chain.

A suffix tree, also known as a position tree or a subword tree, is a tree for storing strings in leaf nodes. Each internal node corresponds to one common prefix. It was first introduced in [1]. Since then it has been widely used in the area of string matching and text editing. The most typical use of it is as an index that supports locating sub-strings of a longer string efficiently.

The sequence  $S$  is mapped to a suffix tree  $T$  whose paths are the suffixes of  $S$ , and whose leaf nodes correspond uniquely to positions within  $S$ . The tree  $T$  has the following properties: 1) An arc of  $T$  may represent any nonempty subsequence of  $S$ . 2) Any common subsequence can be spelled out according to the path from the root to a unique internal node. 3) Each internal node except the root has at least two children. 4) The tree has  $|S|$  leaves, for each nonempty input sequence. Therefore, since each internal node has at least two outgoing edges, the number of nodes is at most  $2|S| - 1$ .

The data structure of a LP-tree is as follows:

**Internal node  $t$ :**  $t.Child$ ,  $t.Ancessor$ ,  $t.Next$ ,  $t.Start$ ,  $t.End$ ,  $t.FirstLeaf$ ,  $t.LastLeaf$ .

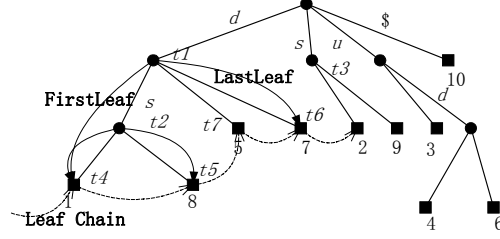


Figure 1. LP tree for “dsuududds\$”

For internal nodes  $t$ ,  $t.Start$  and  $t.End$  indicate the starting position and ending position of the subsequence associated with the branch to  $t$ ,  $t.Child$  stores the pointer to the first child of node  $t$ ,  $t.Ancessor$  stores the pointer to ancestor node of  $t$ ,  $t.Next$  stores the pointer to the next node with the same ancestor node of  $t$ . In order to locate and count the leaf nodes efficiently, all the leaves are linked, forming a leaf chain. For each internal node  $t$ ,  $t.FirstLeaf$  and  $t.LastLeaf$  store the pointer to the first leaf node and the last leaf node of the sub tree  $T(t)$ .

**Leaf  $t$ :**  $t.Ancessor$ ,  $t.Next$ ,  $t.Position$ ,  $t.NextLeaf$

For leaf nodes  $t$ ,  $t.Ancessor$  and  $t.Next$  have the same meaning as that of an internal node,  $t.Position$  is the position of corresponding suffix in  $S$ ,  $t.NextLeaf$  stores the pointer to the next leaf node in the leaf chain.

**Example 2:** The LP tree for the sequence “dsuududds\$” is shown in Figure 1. For the internal node  $t1$ , we have  $t1.Child = \&t2$  where  $\&t2$  denotes the pointer to node  $t2$ ,  $t1.Ancessor = \&root$ ,  $t1.Next = \&t3$ ,  $t1.Start = 1$ ,  $t1.End = 1$ ,  $t1.FirstLeaf = \&t4$ ,  $t1.LastLeaf = \&t6$ . For the leaf node  $t4$ , we have  $t4.Ancessor = \&t2$ ,  $t4.Next = \&t5$ ,  $t4.Position = 1$ ,  $t4.NextLeaf = \&t5$ .

The insertion method for a LP tree is similar to that for a suffix tree. A straightforward way of inserting a suffix is to scan the tree, starting from the root, then to search along a path and stop when the next record in the suffix disagrees. There is also a linear time complexity construction algorithm [9], which starts searching at the lowest possible level with maintaining *suffix links* among non-terminal nodes of tree. These algorithms, efficient in terms of storage and time complexity, can be directly applied on our data structure.

When a node  $t$  is inserted into a LP tree, the relative leaf chain pointers need updating, including the data member *FirstLeaf*, *LastLeaf* and *NextLeaf*. There are two types of operation when a new suffix is inserted. One is linking a new node that represents the suffix to a node directly; another is splitting a node into two nodes and linking the new node to one of them. For the first type of insertion, the update of the leaf chain consists of two steps. Step 1: insert  $t$  into the leaf chain by the operation  $t.NextLeaf = t.Ancessor.LastLeaf.NextLeaf$ , and  $t.Ancessor.LastLeaf.NextLeaf = t$ . Step 2:  $t.FirstLeaf$  and  $t.LastLeaf$  of all ancestor nodes of  $t$  may need updating. During scan the nodes from  $t$  to the root, for each node,

determine whether modification is needed. If no modification is needed for one node, updating is completed. This process can be done easily since there is the pointer  $t.Ancessor$ . For the second type of insertion, which is inserting two nodes into the tree, the updating process seems more difficult, but in fact the operations are similar and time complexity is the same.

**Example 3:** Consider inserting node  $t8$  into the LP tree in Figure 1 with ancestor node  $t2$ .  $t8$  is inserted in the leaf chain between  $t8.Ancessor.LastLeaf = t5$  and  $t8.Ancessor.LastLeaf.NextLeaf = t7$ . Then travel from node  $t2$  to the root. For  $t2$ , the  $t2.lastleaf$  is reset to  $t8$ . For  $t1$ , since  $t1.lastnode$  is not  $t5$ , so it need not be modified. Then the updating process is completed.

Suppose  $A$  is an subsequence, the *local frequency* of  $A$  in any subsequence  $S [sp, ep]$  can be calculated by counting the number of leaf nodes of the sub tree rooted at  $locus(A)$  that satisfies the position restrictions. That is:

$$Lf(A, S[sp, ep]) = |\{leaf(t_A) | sp \leq leaf(t_A).Position \leq ep - |A| + 1\}|$$

Benefiting from maintaining the leaf chain, the number of leaf nodes can be easily and efficiently obtained by a simple traverse in the leaf chain between  $t.Firstleaf$  and  $t.Lastleaf$ . The algorithm for calculating the *local frequency* is given in Algorithm 1.

**Example 4:** Consider the example sequence in example 2. Suppose we are interested in the subsequence “d” and “ds” in “dsuud”, then the *local support* can be calculated as follows:

$$Locus(“d”) = t1, \text{ locus}(“ds”) = t2$$

$$leaf(t1) = \{t4, t5, t6, t7\}, \text{ leaf}(t2) = \{t4, t5\},$$

$$Lf(d, “dsuud”) = |\{leaf(t1) | 1 \leq leaf(t1).Position \leq 5 - |“d”| + 1\}| = |\{t4, t7\}| = 2,$$

$$Lf(ds, “dsuud”) = |\{leaf(t2) | 1 \leq leaf(t2).Position \leq 5 - |“ds”| + 1\}| = |\{t4\}| = 1,$$

---

#### Algorithm 1 $Lf(A, D)$

**Input:** subsequence  $A, D (D=S[sp, ep])$

**Output:**  $Lf(A, D)$ .

$t = locus(A)$

$k = 0$

$tl = t.Firstleaf$

while  $tl \neq t.Lastleaf$

if  $sp \leq tl.Position \leq ep - |subsequence(A)| + 1$  then

$k = k + 1$

$tl = tl.Nextleaf$

end while

Output ( $k$ )

---

## 4.2 Generation of G-LSP Candidates

In the candidates generation phase, all subsequences that are frequent enough in the possible durations, i.e. all  $\langle B, s \rangle$  so that  $Lsupp(B, s) \geq ms$ , are found. Note that, if  $Lsupp(A_{(T)} \rightarrow B(s)) > ms$ , since  $Lsupp(A_{(T)} \rightarrow B(s)) = Lf(AB, s) \leq Lf(B, s) = Lsupp(B, s)$ , there must exists  $Lsupp(B, s) > ms$ . This enable us first find all the frequent subsequences  $B$ , which we term *G-LSP candidates*, and then mine for G-LSPs by searching the records in front of it.

Our strategy for generating G-LSP candidates is: Go through the sequence  $S$  and insert each subsequence  $S [n, \dots, |S|]$  into a LP tree. After a leaf node  $t$  is inserted, only the *local support* of all the patterns stored in the ancestor of  $t$  will be changed. So we need only search the nodes in the path of traveling from node  $t$  to root. For each node  $t_1$  in the travel path, find all the possible subsequences  $D$  such that the *local support* of  $subsequence(t_1)(D)$  is no less than minimal support. These  $\langle subsequence(t_1), D \rangle$  are outputted as G-LSP candidates. The detail of our algorithm for mining G-LSP candidates is given in Algorithm 2.

Let  $(p_1, p_2, \dots, p_n)$  denote the starting positions of subsequence  $A$ , which can be easily calculated from the positions of leaf nodes of sub-tree  $T(A)$ . Note that, given any  $p_m (1 \leq m \leq n)$  and a position  $q (p_{m-1} < q < p_{m+1})$ , the support value of  $A$  in the subsequence from  $p_m$  to the current position  $n$  will not be less than the support value of  $A$  in the subsequence from  $q$  to  $n$ . So we need only consider the subsequences whose starting positions are  $p_m$  instead of scanning all the possible subsequences. It reduces the time complexity dramatically without losing expected candidate.

---

#### Algorithm 2 G-LSP Candidate ( $S, ms$ )

**Input:** sequence  $S$ , minimal support  $ms$ .

**Output:** all G-LSP candidates in sequence  $S$ .

for  $n=1$  TO  $|S|$

insert  $S[n, |S|]$  into a LP tree  $T$ , let the leaf node inserted be  $t$

update corresponding *FirstLeaf*, *LastLeaf* and *Leaf Chain*

for each ancestor node  $t_1$  of  $t$  do

$B = subsequence(t_1)$

for each leaf node  $p$  of  $T(t_1)$

$D = S[p.Position, n]$

$Supp = Lf(B, D) / |D|$

if  $Supp \geq ms$  then output candidate  $\langle B, D \rangle$

end For

end For

end For

---

## 4.3 Discovery of G-LSPs

After a candidate is generated, we use a simple subsequence counting method to generate G-LSPs. Given the candidate  $\langle CB, S[sp, ep] \rangle$ , the detail process is as follows:

1) Retrieve all the positions of the instances of  $CB$  in  $S[sp,ep]$  by going through the leaf nodes of  $subtree(CB)$  in the LP-tree. Denote the results by  $(p_1, p_2, \dots, p_N)$ .

2) Retrieve the subsequences  $s_n = S[p_n - T + |CB|, p_n - 1]$ , and search through  $(s_1, s_2, \dots, s_N)$  to find all subsequence  $A$  that fulfill:

$$Lsupp(A_{(T)} \rightarrow B, S[sp, ep]) = \frac{\sum_{n=1}^N Lf(A, s_n)}{ep - sp + 1} \geq ms$$

$$\text{i.e. } \sum_{n=1}^N Lf(A, s_n) \geq ms(ep - sp + 1)$$

The number of possible  $A$  is at most  $(T - |CB|)(T - |CB| + 1)N/2$ . For each  $A$ ,  $\sum Lf(A, s_n)$  can be easily calculated by searching Leaf (locus( $A$ )) in the LP-tree.

3) For the resulting subsequence  $A$  of step 3, calculate the local frequency of  $A$  in subsequence  $S[sp,ep]$  by going through the leaf nodes of  $subtree(A)$  in the LP-tree. If  $A_{(T)} \rightarrow B(S[sp,ep])$  is a G-LSP,

$$Lconf(A_{(T)} \rightarrow B, S[sp, ep]) = \frac{Lf(A_{(T)} \rightarrow B)}{Lf(A)}$$

$$= \frac{\sum_{n=1}^N Lf(A, s_n)}{Lf(A, S[sp, ep])} \geq mc$$

Then if  $\sum_{n=1}^N Lf(A, s_n) \geq mc \cdot Lf(A, S[sp, ep])$ , the pattern

$A_{(T)} \rightarrow B(S[sp, ep])$  is outputted as a G-LSP.

#### 4.4 Overall Method and Discussions

The overall mining method for G-LSPs is as follows:

1) Go through the sequence  $S$  and insert each subsequence  $S[n, \dots, |S|]$  into a LP tree.

2) After a leaf node  $t$  is inserted, search the nodes in the path of traveling from node  $t$  to root. For each node  $t_1$  in the travel path, find all the possible subsequences  $D$  such that  $|subsequence(t_1)| < T$  and  $\langle subsequence(t_1), D \rangle$  is a G-LSP candidates.

3) For the candidates  $\langle CB, D \rangle$ , search the records in front of the instances of each  $CB$  to find all subsequence  $A$  that fulfill  $A_{(T)} \rightarrow B(D)$  is a G-LSP.

4) Repeat the process 1~3 until the whole sequence has been scanned.

The storage complexity of a LP tree is the same as that of a suffix tree, which is about  $O(|S|)$  [16]. Compared to traditional mining algorithms in which the core operations are pattern generation and tuple counting, our method costs more storage for the LP tree. The new prospects for the data, which result from our algorithm, may justify the added expense. And, it also reduced the mining time expense, which is a crucial factor in the KDD problem.

The expense of both time and storage grow with the length of the sequence. When the sequence is extremely huge, this become a serious problem. On this occasion, a simple “*divide and discover*” strategy can keep the growing under a threshold. The strategy is divide the sequence  $S$  into a set of continuous segments ( $Sp$ ) with the same length so that  $S = S1S2 \dots Sp \dots$ , and use the above G-LSP mining method in each segment. Using this strategy, the LP tree is re-initialized in each segment. Therefore the size of LP tree is restricted, and then the time for mining G-LSPs is also reduced to approximately  $O(|S|)$ .

This strategy makes our approach scaleable. As the tradeoff, a few G-LSPs, of which the time duration is in more than one segments, are neglected. Since the length of the segments won't be too small, the number of missing G-LSPs is very small. So we can treat the missing patterns individually by a simple counting method.

Note that, this strategy is fundamentally different from the naïve sliding window method that we introduced in section 2, because we discover all G-LSPs in each segment, whereas the sliding window methods discover global patterns in the window and can't find all the local patterns.

## 5. EXPERIMENTAL RESULTS

We evaluated the behavior of our problem and the performance of our algorithm on both synthetic data and real data from stock market. The experiment was performed on a machine with 1GHz CPU and 256 megabytes main memory.

### 5.1 Experiments on Synthetic Data

Synthetic dataset was generated using the following parameters:

$L$ : Length of sequence.

$PT$ : Maximal distance, i.e.  $T$  of potential G-LSP  $A_{(T)} \rightarrow B(s)$ .

$PN$ : The number of potential G-LSPs.

$PL1$ : Length of the subsequence  $A$  in potential G-LSP  $A_{(T)} \rightarrow B(s)$ .

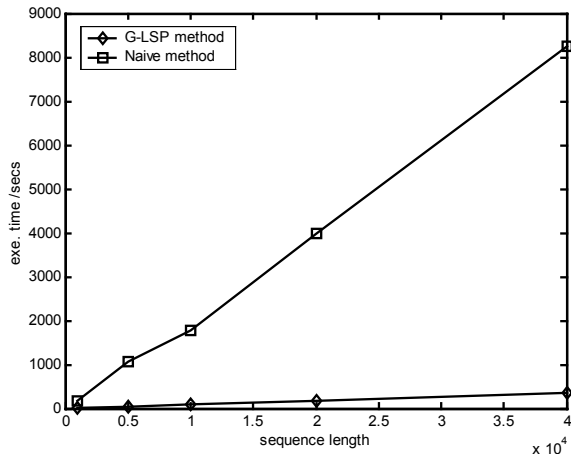
$PL2$ : Length of the subsequence  $B$  in potential G-LSP  $A_{(T)} \rightarrow B(s)$ .

$PD$ : Average duration of potential G-LSPs, i.e.  $|s|$ .

$PS$ : Support of potential G-LSPs

$PC$ : Confidence of potential G-LSPs

The dataset is generated according to the following method. First, generate a sequence with length  $L$ , where each record is randomly selected from the symbol set  $\{a..z\}$ . Second generate both starting position and ending position randomly for each pattern. And the length is normally distributed with  $PD$  means. Third, according to  $PS$ ,  $PC$  and  $PT$ , generate positions of the instances of the corresponding pattern in the sequence, where the positions are uniformly distributed. Then fill in the letters of this pattern at the positions.



**Figure 2. Performance comparison (using dividing strategy)**

In the experiments, minimal support and minimal confidence are set to 80% of the corresponding  $PS$  and  $PC$ . The parameters for generating datasets are given in Table 1 together with the results. Our method successfully discovered all the potential G-LSPs. There are some additional patterns that had also been found. Some of them are parts of the expected patterns, e.g.  $A_{(T)} \rightarrow B$  of expected pattern  $AB_{(T)} \rightarrow C$ . These redundant patterns can be easily phased out in the post-recessing stage. Furthermore, there are a few unexpected patterns that are not related to the patterns we generated. They come from the randomness of the method of generating experimental data.

We also implemented the naïve frequency counting algorithm that scanning every possible subsequence (introduced in section 2), and carried out experiments using generated sequence with varying length for the purpose of performance comparison. In the experiments, both the method introduced in this paper with “*divide and discover*” strategy and the naïve algorithm were used. For the justice of the comparison, we also applied the dividing strategy on the naïve algorithm. Length of segments was set to be 500. The execution time is shown in Figure 2. The results verified the superiority of our method to the naïve one.

## 5.2 Experiments on Real Data

The real data, which were extracted from stock market, are a set of time series collected from 10/1/1997 to 8/1/2000. Each time series is a sequence of real numbers representing daily closing price. First, we use a method called discretizing by clustering windows [10] to discretize it into a symbol sequence in which each symbol represents the series behavior at that time point. The discretizing method is as follows: Given time series  $S = (x_1, \dots, x_N)$ , window  $s_i = (x_i, \dots, x_{i+w-1})$  of width  $w$  and step  $k$  in  $S$  is a contiguous subsequence. Then cluster all the subsequences  $W(S) = \{S_i | i=1, k, \dots, nk \dots, Mk\}$  into sets  $C_1, \dots, C_M$ . For each cluster  $C_h$ , we induct a symbol  $a_h$  from alphabet

**Table 1. Synthetic dataset and experimental results**

Parameters								Discovered Expected Patterns
$L$	$PT$	$PN$	$PL1$	$PL2$	$PD$	$PS$	$PC$	
500	5	2	2	2	50	0.2	0.5	2
500	5	10	1	1	50	0.2	0.5	10
500	10	10	2	2	70	0.2	0.5	10
1K	10	10	1	1	70	0.3	0.8	10
1K	10	20	2	2	70	0.3	0.8	20
10K	10	10	1	2	70	0.3	0.8	10
10K	15	20	2	2	70	0.3	0.8	20

$\Sigma = \{a, \dots, z\}$ . Then the discretized version  $D(S)$  of the sequence  $S$  is obtained by looking for each subsequence  $S_i$  the cluster  $C_{j(i)}$  such that  $S_i \in C_{j(i)}$ , and using the corresponding symbol  $a_{j(i)}$ , i.e.  $D(s) = a_{j(1)}, a_{j(2)}, \dots, a_{j(M)}$ . In our experiment, the window width was set to be 30; and the step was set to be 10.

Due to the space limitation, we show only one dataset in Figure 3, and part of the results on this dataset in Table 2. From the results, we can get some interesting information such as: During the temporal position [20,63], a period of surging was always followed by falls in two days (G-LSP:  $j_{(2)} \rightarrow g$  [20,63]); during [76,84] a period of surging was always followed by another surging in two days (LSP:  $j_{(2)} \rightarrow j$  [76,84]), etc. Such knowledge could benefit the analysis of the stock market with further explanations from domain experts. Furthermore, it is also possible to obtain other novel knowledge through the comparisons of the G-LSPs discovered from the price data of stocks with different business focus.

## 6. CONCLUSIONS

In this paper, we present a problem class: discovery of generalized local sequential patterns (G-LSP), of which the format is “if  $A$  occurs, then  $B$  occurs within time  $T$ ”, in a long temporal sequence. The purpose is to discover patterns that are frequent only in some time durations. This kind of patterns is very common in practice, and the efficient discovery of it will benefit KDD of many real applications.

The problem has a two-dimensional solution space consisting of patterns and temporal features. As we have discussed in section 2, previous algorithms are either inapplicable or have extremely poor time complexity for mining G-LSPs. In this paper, we propose an index structure and a two-phase method for efficiently mining of local patterns. Using this method, prerequisite node searching and counting are accelerated by using leaf node chain during the mining process. All G-LSPs are discovered after one scan of the sequence. In addition, we also propose a “*divide and discover*” strategy that can keep the growing of the storage expense under a threshold and make the time expense scale linearly.

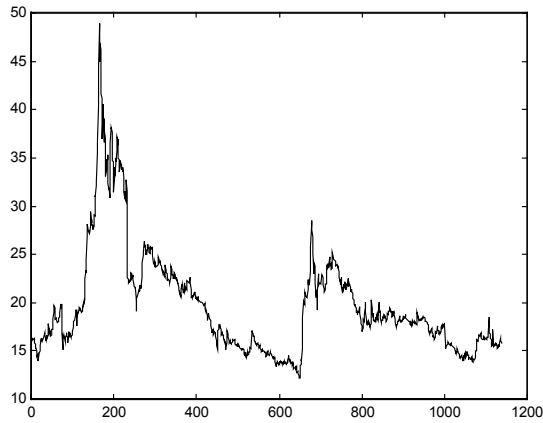
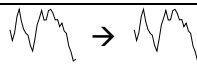
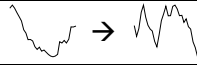
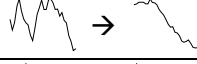



Figure 3. Real data set

Table 2. Experimental results for real data

G-LSP	Meaning of the pattern	Lsupp.	Lconf.
$j_{(2)} \rightarrow j(S[76,84])$		0.44	0.67
$f_{(2)} \rightarrow j(S[32,60])$		0.21	0.86
$j_{(2)} \rightarrow g(S[20,63])$		0.2	0.64
$g_{(2)} \rightarrow f(S[50,59])$		0.30	0.75

We evaluated the behavior of our problem and the performance of our algorithm on both synthetic data and real data. The results correspond with the definition of our problem and reveal a kind of novel knowledge. In addition, experiments using sequences of various lengths verified the superiority of our method to the naïve one.

Discovery of the patterns that are frequent locally is useful in its own right as a tool for the analysis of sequence data. In future, we intend to use it as a subroutine in other KDD applications such as segmentation of sequences [15], exploration by feature, and mining of second order knowledge [11].

## 7. ACKNOWLEDGMENTS

The research has been supported in part of Chinese national key fundamental research program (no. G1998030414) and Chinese national fund of natural science (no. 79990580)

## 8. REFERENCES

[1] P.Weiner. Linear pattern matching algorithms. Conference Record, the IEEE 14th Annual Symposium on Switching and Automata Theory, 1973.

[2] T.G.Dietterich and R.S.Michalski. Discovering patterns in sequences of events. Artificial Intelligence, Vol.25, 1985.

[3] R.Agrawal and R.Srikant. Mining sequential patterns. In Proc. of International Conference On Data Engineering. Taipei, 1995.

[4] R.Srikant, R.Agrawal. Mining sequential patterns: generalizations and performance improvements. The Fifth International Conference on Extending Database Technology, Avignon, France, 1996.

[5] K.Wang and J.Tan. Incremental discovery of sequential patterns. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Montreal, Canada. 1996.

[6] H.Mannila, H.Toivonen, and A.I.Verikamo. Discovering frequent episodes in sequences. The First International Conference on Knowledge Discovery and Data Mining, Canada. 1995.

[7] P.-s.Kam and A.W.-C.Fu. Discovering temporal patterns for interval-based events. Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000), UK. 2000.

[8] Y.Li, X.S.Wang and S.Jajodia. Discovering temporal patterns in multiple granularities. International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining. Lyon, France. 2000.

[9] F.M.McCreight. A Space-economical suffix tree construction algorithm. JACM, Vol 23, No. 2. 1976.

[10] G.Das, K.Lin, H.Mannila, G.Renganathan and P.Smyth. Rule discovery from time series. KDD 98. 1998.

[11] M.Spiliopoulou and J.F.Roddick. Higher order mining: modelling and mining the results of knowledge discovery. Data Mining II - Second International Conference on Data Mining Methods and Databases. 2000.

[12] U.M.Fayyad, G.Piatetsky-Shapiro and P.Smyth. From data mining to knowledge discover: and overview. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press. 1996.

[13] H.Mannila and H.Toivonen. Discovering generalised episodes using minimal occurrences. Second International Conference on Knowledge Discovery and Data Mining (KDD-96). 1996.

[14] J.Roddick and M.Spiliopoulou. A bibliography of temporal, spatial and spatio-temporal data mining research. SIGKDD Explorations, Vol 1, No. 1. 1999.

[15] V.Guralnik and J.Srivastava. Event detection from time series data. Fifth International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA. 1999.

[16] K. Wang, Discovering patterns from large and dynamic sequential data, Special Issues on Data Mining and Knowledge Discovery, Journal of Intelligent Information Systems, 9(1), 8-33, 1997, Kluwer Academic Publishers

[17] S. Ramaswamy, S. Mahajan, A. Silberschatz. On the discovery of interesting patterns in association rules, The VLDB Journal, pages 368-379, 1998

[18] X. Chen, I. Petrounias. An Integrated query and mining system for temporal association rules. The 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000), London, UK. 327-336. 2000.

[19] A. Tansel, N. Ayan. Discovery of association rules in temporal databases. 4th International Conference on Knowledge Discovery and Data Mining (KDD'98) Distributed Data Mining Workshop, NewYork, USA, August 1998.

[20] J. Han, G. Dong and Y. Yin. Efficient Mining of Partial Periodic Patterns in Time Series Database. In Proc. Fifteenth International Conference on Data Engineering, Sydney, Australia. IEEE Computer Society. 106-115. 1999