Mining Freqeunt Episodes for relating Financial Events and Stock Trends

Anny Ng and Ada Wai-chee Fu
Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, Hong Kong
Email: ang,adafu@cse.cuhk.edu.hk

Abstract. It is expected that stock prices can be affected by the local and overseas political and economic events. We extract events from the financial news of Chinese local newspapers which are available on the web, the news are matched against stock prices databases and a new method is proposed for the mining of frequent temporal patterns.

1 Introduction

In stock market, the share prices can be influenced by many factors, ranging from news releases of companies and local politics to news of superpower economy. We call these incidences **events**. We assume that each event is of a certain **event type** and each event has a time of occurrence, typically given by the date that the event occurs or it is reported. Each "event" therefore corresponds to a time point. We expect that events like "the Hong Kong government announcing deficit" and "Washington deciding to increase the interest rate", may lead to fluctuations in the Hong Kong stock prices within a short period of time. When a number of events occur within a short period of time, we assume that they possibly have some relationship. Such a period of time can be determined by the application experts and it is called a **window**, usually limited to a few days. Roughly speaking, a set of events that occur within a window is called an **episode instance**. The set of event types in the instance is called an **episode**.

For example, we may have the following statement in a financial report: "Telecommunications stocks pushed the Hang Seng Index 2% higher following the Star TV-HK Telecom and Orange-Mannesmann deals". This can be an example for an episode, in which all the four events, "telecommunication stocks rise", "Hang Seng Index surges" and the two deals of "Star TV-HK Telecom" and "Orange-Mannesmann", all happened within a period of 3 days. If there are many instances of the same episode it is called a *frequent episode*. We are interested to find frequent episodes related to stock movements. The stock movement need not be the last event occurring in the episode instance, because the movement of stocks may be caused by the investors' expectation that something would happen on the following days. For example, we can have a news report saying "Hong Kong shares slid yesterday in a market burdened by the fear of possible United States interest rates rises tomorrow". Therefore we do not assume an ordering of the events in an episode.

From the frequent episode, we may discover the factors for the fluctuation of stock prices. We are interested in a special type of episodes that we call **stock-episodes**, it can be written as "$\langle e_1,\ e_2,\ ...\ e_n\ (t \text{ days})\rangle$", where the e_1, e_2, ... e_n are event types and at least one of the events should be the event of stock fluctuation. An instance for this stock-episode is an instance where the events of the event types e_1, ... e_n appear in a window of t days. Since we are only concerned with stock-episodes, we shall simply refer to stock-episodes as episodes.

1.1 Definitions

Let $E = \{E_1, E_2, ..., E_m\}$ be a set of **event types**. Assume that we have a database that records events for days 1 to n. We call this a **event database**, we can represent this as $DB =< D_1, D_2, ..., D_n >$, where D_i is for day i, and $D_i = \{e_{i1}, e_{i2}, ..., e_{ik}\}$, where $e_{ij} \in E$ ($j \in [1, k]$), This means that the events that happen on day i have event types $e_{i1}, e_{i2}, ..., e_{ik}$. Each D_i is called a **day-record**. The day records D_i in the database are consecutive and arranged in chronological order, where D_i is one day before D_{i+1} for all $n-1 \geq i \geq 1$. $P = \{e_{p1}, e_{p2}, ..., e_{pb}\}$, where $e_{pi} \in E$ ($i \in [1, b]$), is an **episode** if P has at least two elements and at least one e_{pj} is a stock event type. We assume that a **window size** is given which is x days, this is used to indicate a consecutive sequence of x days. We are interested in events that occur within a short period as defined by a window. If the database consists of m days and the window size is x days, there are (m) windows in the database: The first window contains exactly days $D_1, D_2, ..., D_x$ The i-th window contains $D_i, D_{i+1}, ...$, with up to x days. The second last window contains D_{m-1}, D_m, and the last window contains only D_m.

In some previous work such as [6], the frequency of an episode is defined as the number of windows which contain events in the episode. For our application, we notice some problem with this definition: suppose we have a window size of x, if an episode occurs in a single day i, then for windows that start from day $i-x+1$ to windows starting from i, they all contain the episode, so the frequency of the episode will be x. However, the episode actually has occurred only once. Therefore we propose a different definition for the frequency of an episode.

Definition 1. *Given a window size of x days for DB, and an episode P, an* **episode instance** *of P is an occurrence of all the event types in P within a window W and where the record of the first day of the window W contains at least one of the event types in P. Each window can be counted at most once as an episode instance for a given episode.*

The **frequency of an event** *is the number of occurrences of the event in the database. The* **support** *or the* **frequency** *of an episode is the number of instances for the episode. Therefore, the frequency of an episode P is the number of windows W, such that W contains all the event types in P and the first day of W contains at least one of the event types in P. An episode is a* **frequent episode** *if its frequency is $>=$ a given* **minimum support threshold**. □

Problem definition: Our problem is to find all the frequent episodes given a event database and the parameters of window size and minimum support threshold.

Let us call the number of occurrences of an event type a in DB the **database frequency** of a. Let us call the number of windows that contain an event type a the **window frequency** of a. The window frequency of a is typically greater than the database frequency of a since the same occurrence of a can be contained in multiple windows. We have the following property.

Property 1. For any episode that contains an event a, its frequency must be equal to or less than the window frequency of a. That is, the upper limit of the frequency of an episode containing a is the window frequency of a.

Lemma 1. *For a frequent episode, a subset of that episode may not be frequent.*

Proof: We prove by giving a counter example to the hypothesis that all subsets of a frequent episode are frequent. Suppose we have a database with 7 days, and the window size is 3, the records D_1 to D_7 are: $\{b\}, \{a, c\}, \{b\}, \{d\}, \{b\}, \{c, a\}, \{d\}$, respectively. If the threshold is 3, then $< abc >$, has 3 occurrences and is a frequent episode, while $< ac >$, which is a subset of $< abc >$, has only 2 occurrences and is not a frequent episode. □

1.2 Related Work

The mining of frequent temporal patterns has been considered for sales records, financial data, weather forecast, and other applications. The definitions of the patterns vary in different applications. In general an episode is a number of events occurring within a specific short period of time. The restriction of the ordering of events in an episode depends on the applications. Previous related research includes discovering sequential patterns [1], frequent "episodes" [6], temporal patterns [4] and frequent patterns ([3, 2]). In [6] an episode, defined as the "partially ordered" events appearing close together, is different from our definition of stock-episode. Some related work focus on stock movement [5], but we would like to relate financial events with stock movement.

When we deal with the events which last for a period of time, we may consider the starting time and ending time of the events as well as their temporal relations, such as overlap and during. [4] discovers more different kinds of temporal pattern.

[9] discovers frequent sequential patterns by using a tree structure. [6] finds the frequent series and parallel episodes in a sequence of point-based events. An episode is a partially ordered of events occurring close together. An episode X is a subepisode of another episode Y if all events in X are also contained in Y and the order of events in X is the same with that in Y. The frequency of an episode is the number of windows containing the episode. Note that this definition is different from ours, since it allows the same episode instance to be counted multiple times when multiple windows happen to contain the instance.

Most of the algorithms introduced in the above are based on Apriori Algorithm [1]. Our definition of frequent episodes does not give rise to the subset closure property utilized in these methods. [3] provides a fast alternative to find the frequent pattern with a frequent pattern tree (FP-tree), which is a kind of prefix tree. However, the FP-tree is not designed for temporal pattern mining. There is some related work in applying the technique to mine frequent subsequences in given sequences [8], but the problem is quite different from ours.

2 An Event Tree for the database

The method we propose has some similarity to that in [3]. We use a tree structure to represent the sets of event types with paths and nodes. The process is comprised of two phrases: (1) Tree construction and (2) Mining frequent episodes.

The tree structure for storing the event database is called the **event tree**. It has some similarity to the FP-tree. The root of the event tree is a null node. Each node is labeled by an event type. Each node also contains a **count**, and a **binary bit**, which indicates the *node type*.

Before the event tree is built, we first gather the frequencies of each event type in the database DB. We sort the event types by descending frequencies. Next we consider the windows in the database. For each window,

1. Find the set F of event types in the first day, and the set R of event types in the remaining days. F and R are each sorted in descending database frequency order.
2. Then the sorted list from F and that from R are concatenated into one list and inserted into the event tree. One tree node corresponds to each event type in each of F and R. If an event type is from F, the binary bit in the tree node is 0, if the event type is from R, the binary bit in the tree node is 1. Windows with similar event types may share the same prefix path in the tree, with accumulated count. Hence a path may correspond to multiple windows. If a new tree node is entered into the tree, the count is initialized to 1. When an event type is inserted into an existing node, the count in the node is incremented by 1.

In the event tree, each path from the root node to a leaf node is called a **window path**, or simply a path, when no ambiguity can arise. The event tree differs from an FP-tree in that each window path of the tree is divided into two parts. There is a cut point in the path so that the nodes above the cut point has binary bit set to 0. This is called the **firstdays part** of the path. The second part of the path, with binary bits of 1, is called the **remainingdays part** of the path.

There is a header table that contains the event types sorted in descending order of their frequencies. Each entry in the header table is the header of a linked list of all the nodes in the event tree labeled with the same event type as the header entry. Each time a tree node x is created with a label of event type e, the node x is added to the linked list from the header table at entry e. The linked list therefore has a mixture of nodes with binary bits of 0 or 1.

The advantage of the event tree structure is that windows with common frequent event types can likely share the same prefix nodes in the event tree. In each of the firstdays part and the remainingdays part, the more frequent the event is, the higher level the event node is in so as to increase the chance of reusing the existing nodes.

Before building the tree, we can do some pruning based on event type frequencies. Those event types with window frequencies less than the minimum support threshold are excluded from the tree because the event types will not appear in the frequent episodes with the reason stated in Property 1. Once an event type is excluded, it will be ignored whenever it appears in a window. This helps us to reduce the size of tree and reduce the chance of including non-frequent episodes.

Strategy 1 *We remove those events with the window frequencies less than the minimum support threshold before constructing the tree,*

Strategy 2 *In counting the windows for the frequency of an episode, each window can be counted at most once. If an event type appears in the first day and also in the remaining days of a window simultaneously, the effect on the counting is the same as if the event type appears only in the first day. Therefore, for such a window, only the occurrence of the event type in the first day will be kept and that in the remaining part of the window is/are removed.*

Example 1: Given an event database as shown in Figure 1(a), suppose the window size is set to 2 days and the minimum support is set to 5, the event database is first scanned to sum up the frequencies of each event type in the database and also the window frequencies, which are $< a:4, b:5, c:3, d:3, m:2, x:3, y:2, z:2 >$ and $< a:6, b:7, c:5, d:6, m:4, x:5, y:4, z:3 >$, respectively. Thus the frequent event types are a, b, c, d, x since their window frequencies are at least the minimum support. The frequent event types are sorted in the descending order of their database frequencies and the ordered frequent event types are $< b, a, c, d, x >$.

Day	Events
1	a, b, c
2	y, m, b, d, x
3	a, c
4	a, b
5	z, m, y, d
6	b, x, c
7	d, a, b
8	x, z

(a)

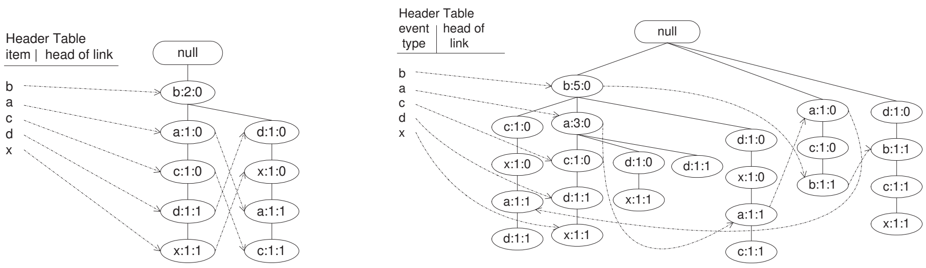
Window No.	Days included	Event-set pairs
1	1,2	$< (b, a, c), (d, x) >$
2	2,3	$< (b, d, x), (a, c) >$
3	3,4	$< (a, c), (b) >$
4	4,5	$< (b, a), (d) >$
5	5,6	$< (d), (b, c, x) >$
6	6,7	$< (b, c, x), (a, d) >$
7	7,8	$< (b, a, d), (x) >$
8	8	$< (x), () >$

(b)

Fig. 1. An event database and the corresponding windows

Next a null root node is created. The event database is then scanned for the second time to read the event types in every 2 days for inserting the windows' event types into the tree. Keeping only the frequent event types and excluding the duplicate event types, the first window can be represented by $<(b,a,c),(d,x)>$, in which the first round brackets consists of the event types in the first day of window while the second round brackets consists of the event types in the remaining days of the window (the second day of the window in this example). Both event lists are sorted in decreasing window frequency order. We call $<(b,a,c),(d,x)>$ the **event-set pair** representation for the window. A first new path is built for the first window with all counts initialized to one. The nodes are created in the sorted order and the types of the nodes b, c and a are set to 0 while that of nodes d and x are set to 1.

The window is shifted one day lower and one more day of event types in the database are read to get the second window. The event types are sorted and the second window is $<(b,d,x),(a,c)>$. The tree after inserting the path for the second window is shown in Figure 2. Each tree node has a label of the form $<E:C:B>$ where E is an event type, C is the count, and B is the binary bit. In this figure, the dotted lines indicates the linked list originating from items in the header table to all nodes in the tree with the same event type.



(a) (b)

Fig. 2. (a) The tree after inserting the first two windows. (b) A rough structure of the final tree constructed in Example 1.

The remaining windows are inserted to the tree in the similar way. The rough structure of the final tree constructed is shown in Figure 2(b). (Note that some dotted lines are missing in the figure for clarity.)

3 Mining frequent episodes with the event tree

Our mining process is a recursive procedure applied to each of the linked list kept at the header table. Let the event types at the header table be $h_0, h_1, ...h_H$, in the top-down ordering of the table. We start from the event type h_H at the bottom of the header table and traverse up the header table. We have the following objective in this recursive process:

Objective A: *Our aim is that when we have finished the processing of the linked list for* h_i*, we should have mined all the frequent episodes that contain event types* $h_i, h_{i+1}, ..., h_H$.

Suppose we are processing the linked list for event type h_i. $\{h_i\}$ is called a **base event set** in this step. We can examine all the paths including the event type h_i from the event tree by following the linked list. Let us call the set of these paths P_i. These paths will help us to find the frequencies of episodes containing event type h_i. We have the following objective:

Objective B: From the paths in P_i, we should find all frequent episodes that contain h_i but not any of h_{i+1}, ..., h_H.

The reason why we do not want to include h_{i+1}, ... h_H is that frequent episodes containing any of h_{i+1}, ..., h_H have been processed in earlier iterations. Let us call the set of all frequent episodes in DB that contain h_i but not any of h_{i+1}, ... h_H, the set X_i.

We break up the Objective B into two smaller objectives:

Objective B1: *From the paths in* P_i*, we would like to find all frequent episodes in* X_i *of the form* $\{a\} \cup \{h_i\}$*, where* a *is a single event type.*

Objective B2: *From the paths in* P_i*, we would like to form a database of paths* DB' *which can help us to find the set* S_i *of all frequent episodes in* X_i *that contains* h_i *and at least two other event types.* DB' *is a* **conditional database** *which does not contain* $\{h_i\}$ *such that if we concatenate each conditional frequent episode in* DB' *with* h_i*, the resulting episodes will be the set we want.*

With DB' we shall build a conditional event tree T' with its header table in a similar way as the first event tree. Therefore, we can repeat the mining process recursively to get all the conditional frequent episodes in T'.

Now we consider how we can get the set of paths P_i, and from there obtain a set of **conditional paths** C_i in order to achieve Objectives B1 and B2. Naturally we examine the linked list for h_i, and locate all paths that contain h_i. Let us call the event types $h_{i+1}, ...h_H$ **invalid** and the other event types in the header table **valid**. A node labeled with an invalid(valid) event type is invalid (valid). Suppose we arrive at a node x in the linked list, there are two possibilities

1. If the node x (with event type h_i) is at the firstdays part (the binary bit is 0), we first visit all the ancestor nodes of x and include all the nodes in our **conditional path prefix**. We perform a *depth-first search* to visit all the sub-paths in the sub-tree rooted under event node x, each such path has a potential to form a conditional path. Only valid nodes are used to form paths in P_i. Note that the nodes in the firstdays part of the path below event x will be invalid and hence ignored.
2. If the node x (with event type h_i) is in the remainingdays part, we simply traverse up the path and ignore the subtree under this node. This is because all the nodes below x will be invalid. Invalid nodes may appear above x and they are also ignored.

For example, when we process the left most $< d : 1 : 1 >$ *node in the tree in Figure 2(b), we traverse up the tree, include all nodes except for* $< x : 1 : 0 >$, *since it is invalid. When we process the left most $< c:1:0 >$ node in the tree in Figure 2(b), we traverse up to node $< b:5:0 >$, and then we do a depth first search. We ignore the nodes $< x:1:0 >$ below it, and include $< a:1:1 >$ but not $< d:1:1 >$.* Note that the downward traversal can be stopped when the current node has no child node or we have reached an invalid node.

Let us represent a path in the tree by $< (e_1 : c_1, e_2 : c_2, ..., e_p : c_p), (e'_1 : c'_1, ..., e'_q : e'_q) >$, where e_i, e'_j are event types, c_i, c'_j are their respective counts, e_i are event types in the firstdays part, and e'_j are from the remainingdays part. Consider a path p that we have traversed in the above. We effectively do a few things for p:

- **Step (1)**: Remove invalid event types, namely, $h_{i+1}, ...h_H$.
- **Step (2)**: Adjust counts of nodes above h_i in the path to be equal to that of h_i
- **Step (3)**: If h_i is in the firstdays part, then move all event types in the remainingdays part to the firstdays part
- **Step (4)**: Remove h_i from the path.

The resulting path is a conditional path for h_i. After we have finished with all nodes in the linked list for h_i, we have the complete set of conditional paths C_i for h_i. For example, for Figure 2, the conditional paths for x are $< (d:1), (b:1, c:1) >, < (b:1, c:1, a:1, d:1), (\phi) >, < (b:1, a:1, c:1), (d:1) >, < (b:1, a:1, d:1), (\phi) >, < (b:1, d:1, a:1, c:1), (\phi) >$.

The set C_i forms our conditional database DB'. It helps us to achieve both Objectives B1 and B2. For Objective B2, we first determine those event types in DB' with a **window frequencies** which satisfies the minimum threshold requirement. This can help us to prune some event types when constructing the conditional event tree T'. The window frequency of e is the sum of the counts of nodes in C_i with a label of e.

For Objective B1, we need to find the single event types which when combined with h_i will form a frequent episode. For locating these event types we use the **first-part frequency** for event types. The first-part frequency of an event type e in the set of conditional paths C_i is the sum of the counts in the nodes with label e in C_i that are in the firstdays part.

In the above we describe how we can form a conditional database DB' with a base event set $\alpha = \{h_i\}$, and a conditional event tree T' can be built from DB'. In the header table for T', the event types are sorted in descending order of the database frequencies in DB'. Event types in the conditional paths in DB' are then sorted in the same order at both the firstdays part and the remainingdays part before they are inserted into T'.

We apply the mining process recursively on this event tree T'. T' has its own header table and we can repeat the linked list processing with each entry in the header table. When we build another conditional event tree T'' for a certain header h'_j for T', the *event base set* is updated as $h'_j \cup \alpha$. This means that frequent episodes uncovered from T'' are to be concatenated with $h'_j \cup \alpha$ as the resulting frequent episodes.

Strategy 3 *When a conditional event tree contains only a single path, the frequent episodes can be generated directly by forming the set S_1 of all possible subsets of the event types in the firstdays part of path, and then the set S_2 of all possible subsets of the event types in the remainingdays part. Any element of S_1 with the event base set is a possible frequent episode. The union of any element of S_1 and any element of S_2 and the event base set is also a possible frequent episode. And the frequency of such a episode is the minimum among the event types in episode.*

By the way we construct a conditional event tree, if a path contains event types in the remainingdays part, those event types corresponds to windows which contains some episode e with h_i in the remainingdays part. For such windows to be counted for the episode e, there must be some event type in e that occurs in the firstdays part. Therefore when we form episode with an element in S_2 we must combine with some element in S_1.

4 Performance evaluation

To evaluate the performance of the proposed method, we conduct experiments on an Sun Ultra 5_10 machine running SunOS 5.8 with 512 MB Main Memory. The programs are written in C. Both synthetic and real data sets are used.

Synthetic data: The synthetic data sets are generated from a modified version of the synthetic data generator in [1]. The data is generated with the consideration of overlapping windows. That is, with the window size of x days, the program will consider what data it has generated in the previous $x-1$ days, in order to choose the suitable event types for the x-th day to maintain the target frequencies of the frequent episodes. The data generator takes six main parameters as listed in the following table:

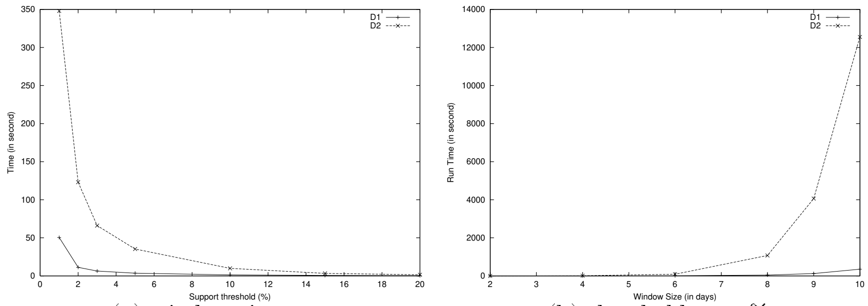
Parameter	Description	Values
$\|D\|$	Number of days	1K, 2K, 3K
$\|T\|$	Average number of events per day	10, 20
$\|I\|$	Average size of frequent episodes	3, 5
$\|L\|$	Number of frequent episodes	1000
M	Number of event types	100 - 1000
W	Window size	2 - 10

Four datasets with different parameter settings as shown in the following table are produced. With D1 and D2, we vary the thresholds and window sizes. With D2 to D4 we vary the number of days and event types.

Dataset Name	Dataset	$\|T\|$	$\|I\|$	$\|D\|$	$\|M\|$
D1	T10.I3.M500.D1K	10	3	1K	500
D2	T20.I5.M1000.D3K	20	5	3K	1000
D3	T20.I5.M100.D2K	20	5	2K	200
D4	T20.I5.M500.D2K	20	5	2K	500

In our implementation, we used linked lists to keep the frequent episodes, one list for each episode size. Each of these lists is kept in an order of decreasing frequencies for a ranked display to the user at the end of the mining. We measure the run time as the total execution time of both CPU time and I/O time. The run time in the experiment include both tree construction and mining. Each data points in graphs are the mean time of several runs of the experiment.

The run time decreases with the support threshold as shown in Figure 3 (a). As the support threshold increases, less frequent events are found and included in the subsequent conditional trees and much less time are required to find the frequent event types in the smaller conditional trees.



(a) window size = 3 (b) threshold = 20%

Fig. 3. Performance of synthetic datasets D1 and D2

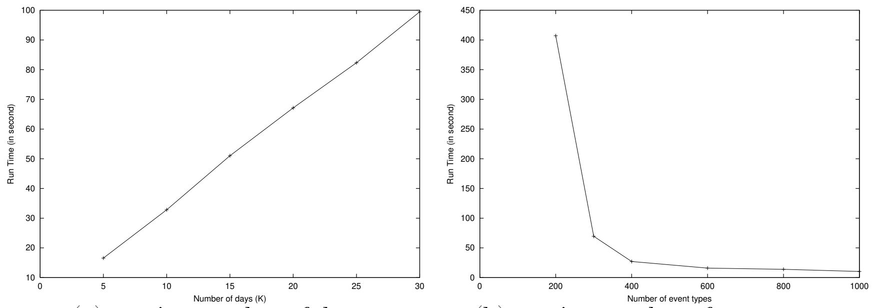
Figure 3(b) shows the effect of different window sizes on the run time. The datasets D1 and D2 are used and the experiment run under threshold fixed to 20%. When the window size increases, the execution time increases because more items are included in window and paths of trees. The parameters of D2 are greater than D1 and therefore the sizes of the initial tree and the conditional trees are larger. So the run time for D1 is much larger than that for D2 when the window size is greater than 8 days.

To study the effect of the number of days in datasets on the execution time, the experiment on dataset D2 is conducted. The support threshold and the window size are set to 10% and 3 days respectively. The result in Figure 4(a) shows that the execution time increases linearly with the number of days.

The effect of the number of event types on the execution time is also investigated. The dataset D2 is used and the support threshold is set to 10% with 3 days of window size. The result is shown in Figure 4(b).

The curve falls exponentially as the number of event types increases. When the number of items is decreased, the distribution of event types are more concentrated and the frequencies of the event types are higher. Therefore less events are pruned when constructing the conditional trees and the run time is longer.

Real data: The real data set is the news event extraction from a internet repository of a number of local newspapers, more details of which is reported in [7]. It contains 121 event types and 757 days. For example, Cheung Kong stock goes up is an event type. An event for an event type occurs when it is reported in the collected news. In addition we have collected stock data from the Datastream International Electronic Databse, we have retrieved Dow Jones

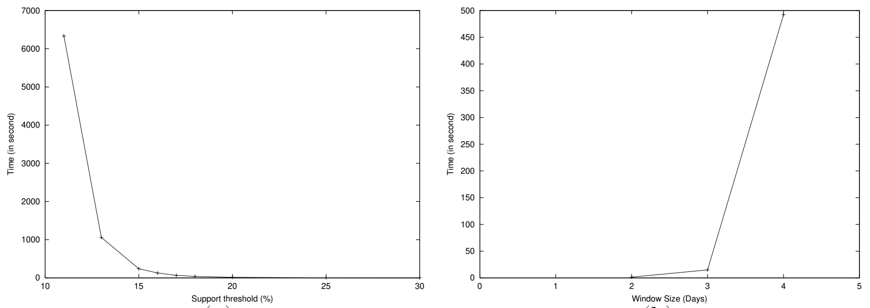


(a) varying number of days (b) varying number of event types

Fig. 4. Synthetic dataset D2 with window size = 3 and threshold = 10%.

industrial average, Nasdaq Composite Index, Hang Seng Index Future, Hang Seng Index, and prices of 12 top local companies for the same period of time.

The performance using the real dataset with different support thresholds and window sizes are shown in Figure 5 (a) and (b). In Figure 5 (a), the window size is set to 3 days. The execution time is rapidly decreased with the threshold above 15%. It is because the supports of half of the most frequent events are close together, when the threshold is below 17%, the pruning power in forming conditional trees is weak.



(a) (b)

Fig. 5. Real dataset with (a) window size = 3 days. (b) support threshold = 20%.

The performance on varying the window sizes are shown in Figure 5(b) with the threshold equal to 20%. The execution time increases with the window size steeply. The run time with a window size of 5 days is too long (> 30000 seconds) and is not shown in graph. When the window size is large, the tree paths are longer and include more items. As the supports of the items are close together, the subsequent conditional trees nearly include all event types from the original trees and the sizes of conditional trees cannot be reduced. Thus the mining time increases with the window size.

Table 1. Some of the results mined with threshold = 15% and window size = 3 days.

Episode	Support
Nasdaq downs, PCCW downs	151
Cheung Kong ups, Nasdaq ups	129
Cheung Kong Holdings ups, China Mobile Ups	128
Nasdaq ups, SHK Properties flats, HSBC flats	178
Cheung Kong ups, SHK Properties flats, HK Electric flats	178
China Mobile downs, Nasdaq downs, HK Electric flats	178
China Mobile downs, Heng Sang Index downs, HSBC flats	135
US increases interest rate, HSBC flats, Dow Jones flats	100

Real Dataset Results Interpretation: Since the frequencies of the events obtained from newspapers are much less than the events of stock price movement, we have set the threshold to 15% to allow more episodes including the newspaper events to be mined. We have selected some interesting episodes mined with threshold set to 15% and window size set to 3 days in Table 1. We notice some relationship between Nasdaq and PCCW, a telecom stock. We see that Nasdaq may have little impact on SHK Properties (real estate), or HSBC (banking).

Acknowledgements: This research was supported by the RGC (the Hong Kong Research Grants Council) grant UGC REF.CUHK 4179/01E.

References

1. R. Agrawal and R. Srikant. *Mining sequential patterns.* 11th International Conf. On Data Engineering, March 1995.
2. M.S. Chen, J.S. Park, and P.S. Yu. *Efficient Data Mining for Path Traversal Patterns.* IEEE Transactions on Knowledge and Data Engineering, March/April 10:2, 1998.
3. J. Han, J. Pei, and Y. Yin. *Mining Frequent Patterns without Candidate Generation.* SIGMOD, 2000.
4. P. S. Kam and A. W. C. Fu. *Discovering Temporal Patterns for Interval-Based Events.* Proc. Second International Conference on Data Warehousing and Knowledge Discovery, 2000.
5. H. Lu, J. Han, and L. Feng. *Stock Movement Predication and N-Dimensional Inter-Transaction Association Rules.* Proc. of SIGMOD workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98), 1998.
6. H. Mannila and H. Toivonen. *Discovering generalized episodes using minimal occurrences.* 2nd International Conf. On Knowledge Discovery and Data Mining, August 1996.
7. Anny Ng and K.H. Lee. *Event Extraction from Chinese Financial News.* International Conference on Chinese Language Computing (ICCLC), 2002.
8. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth.* Proceedings of the 12th IEEE International Conference on Data Engineering, 2001.
9. P. C. Wong, W. Cowley, H. Foote, and E. Jurrus. *Visualizing Sequential Patterns for Text Mining.* Proceedings IEEE Information Visualization, October 2000.