Information Extraction and Data Mining from Chinese Financial News

NG Anny

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

從經濟新聞抽取中文訊息及採集股市趨勢

吳媛

香港中文大學

計算機科學與工程學課程

哲學碩士論文

2002年8月

摘要

如果能掌握股市的走勢，這肯定能幫助預測股市。 股票價格除了受其他股票影響外，也會因本地及外國的政治及經濟事件而波動。 本論文旨在探討股市走勢與過去事件的關係。除了從公眾資源蒐集過去的經濟數據， 我們建立了從本地中文經濟新聞中辨認及抽取當日發生的事件的系統，然後根據所得的事件及股票價格，再找尋股市模式。

從中文報章抽取信息之前， 首要任務是把中文文本的連續字串切分成詞的序列，即自動分詞。已切分的文本的其中一項用途便是自動摘文。 我們建立自動摘文系統，利用遺傳算法抽取符合選擇機制的重要句子作為摘要。 此系統基於句子的關鍵字出現次數，在文本的位置，以及與其他句子的相似度及距離等因素， 決定摘要句子。

我們把新聞事件及股票價格互相匹配，然後採集常見股市及經濟事件的模式，藉此了解經濟事件對股市的影響。為了能更有效規劃此問題，我們提出頻繁情節的新定義。這個新定義沒有"頻繁情節的子情節必定也是頻繁情節"的特性。 我們提出新的採集頻繁情節的方法，並應用於由報章所抽取的事件建立的資料庫，找出頻繁情節。實驗證明此方法能有效採集頻繁情節及可應付大型資料庫。

我們也讓用戶設定對採集期望結果的限制, 以加強採集系統。此採集方式只要求輸入要採集最頻繁情節的數量， 免除因決定最小支持度的煩惱。我們也提出了方法， 實驗顯示這兩個方法也能有效採集頻繁情節。

Information Extraction and Data Mining from Chinese Financial News

Submitted by

NG Anny

for the degree of Master of Philosophy in

Computer Science and Engineering

at the Chinese University of Hong Kong

in

August 2002

Abstract

It is useful if we could predict and formulate the stock market trends. Apart from the inter-influence among stocks themselves, stock prices are affected by local and overseas political and economic events. We are interested in the discovery of factors related to stock movements from past data. For the past stock prices, they can be obtained from open sources directly. In order to get the events happening each day, we have built a system to extract events from the financial news of local Chinese newspapers. The events extracted and the stock prices databases are then analyzed to find patterns of the stock market.

To extract information from the Chinese newspapers, the news articles are first segmented into words. After segmentation, information can be extracted from the Chinese text, such as in text summarization. A summarizer is built to get the most important sentences from the financial news by using genetic algorithm, which searches sentences to satisfy a set of heuristic rules and sentences satisfying the rules as a summary are extracted. In addition to the frequencies of keywords and the locations of sentences, the proximities and distances between summary sentences are considered in determining the summary sentences.

To understand better the relationship between some financial events and the actual stock market, the news are matched against stock price databases for the mining of frequent temporal patterns from the stock market and financial news events. We propose a new definition of frequent episodes which can better formulate our problem. This new definition does not have the property that the subset of a frequent episode must be frequent. We propose a new mining method for this problem which we show by experiment to be highly efficient and scalable. We applied our method to a real dataset collected from local financial news. We show that the method is effective in discovering interesting financial patterns.

We enhance the system by allowing users to set constraints on the expected results. The first enhancement is mining the N-most frequent episodes, in which only N is specified instead of the support threshold. This avoids the difficult situation for a novice user to choose a suitable support threshold. To further facilitate users to get episodes containing the desired events, we also propose the method of mining frequent episodes with event constraints. With enhancements, we can also discover those episodes with low supports. Experiments results show that the performance of the methods is efficient.

Acknowledgments

I would like to thank my supervisors, Prof. Kin-Hong Lee and Prof. Ada Wai-chee Fu for inspiring my interests in data mining. Without their support, specialized advice and the helpful discussions with them, the thesis could not be finished.

It is my pleasure to have Prof. Kwong-sak Leung and Dr. Yiu-sang Moon as my thesis markers. In addition, I would like to thank Prof. Chin Lu for her graciously consent to be my external marker. Thanks to their valuable opinion and comments, my thesis and research work can be further improved.

I am grateful to my friends for their encouragement and support which help me to get though the tough moments, and for their help and care which give me a memorable postgraduate life.

I would also like to thank my family for their love and support.

Content

List of Figures

List of Tables

Chapter 1

Introduction

Information extraction (IE) is the analysis of unrestricted text to extract certain specific information. It differs from information retrieval (IR), the traditional technique of processing text information. While IE gets the facts out of documents, IR searches and gets the sets of relevant documents.

When the target information is related to Hong Kong's local issues, we have to deal with Chinese documents in order to get more information. The web, a virtual library which is rich in up-to-date information, provides a convenient way to access documents.

Information extraction becomes more important in recent years. The main goal of information extraction is to produces databases with extracted facts, which are then further analysed using **data mining** techniques to discover meaningful patterns and trends.

There are many areas in data mining, one of which is frequent pattern mining. Frequent pattern, which is a kind of association rules, is a set of frequent items, the number of appearances of which is above a pre-assigned threshold. Frequent pattern mining plays a significant role in mining episodes, sequential pattern, association and correlation.

1.1 Problem Definition

With the rapid development of the web, there are uncountable and wide ranges of documents available. However, users may spent a lot of time to browse every document retrieved even though a powerful search engine is used. At the same time, we are interested in the relationship between the financial events and the actual stock market. We extract the relevant information from the Chinese news documents on web, then further analyses the information with data mining techniques.

In the first part, we notice there are many documents available on the web and it is inefficient to obtain the target documents from the list of documents retrieved. Since there have been rather considerable amount of research on summarization in English documents, we propose a method to automatically generate summaries for Chinese documents. The summaries provide precise and brief descriptions for users to facilitate reading.

In the second part, we attempt to discover the frequent financial episodes, which is a set of financial events appearing close together and their frequencies meet the support threshold. We extract events from the financial news of local Chinese newspapers and insert the events to the event database. We propose a new method to mine the frequent episodes, which can be referred for prediction. It would be more user-friendly to specify the number of the most frequent episodes to be mined. We introduce a method to mine the N-most frequent episodes. In reality, user always want to obtain the episodes including specific events. We describe a method to mine the frequent episode with event constraints.

We believe that the frequency distribution in database affects the setting of threshold and the number of most frequent episodes to be mined (N) in order to obtain the desired episodes. The three methods proposed are able to cope with different types of database and satisfy the needs of users.

1.2 Thesis Organization

There are six chapters following this introduction chapter. The first two chapters involves information extraction, while the following three chapters discuss the data mining methods to discover the knowledge from the information extracted.

Chapter 2 presents a method which applies genetic algorithm to automatically generate a summary for the Chinese new document obtained from web. Chapter 3 proposes a method to extract events from the Chinese financial newspapers. We construct an event database containing the events and the record of the previous stock movements which is obtained directly from open source.

Chapter 4 gives a survey on mining frequent pattern. It also presents two methods to discover the frequent episodes from the event database built in Chapter 2. In Chapter 5, we improve the methods in Chapter 4 to provide a more practical way that mines the N-most frequent episodes. Chapter 6 imposes more restrictions on the results and proposes a method to mine the frequent episodes with event constraints. Finally we draw a conclusion in Chapter 7.

Chapter 2

Chinese Text Summarization Using Genetic Algorithm

Genetic algorithm has been widely used in different domains to search for the optimal solution. In this chapter, a new method applying genetic algorithms on summarizing the Chinese text is proposed. The summarizer gets the most important sentences from the financial news determined by a genetic algorithm, which searches sentences to satisfy a set of heuristic rules and extract the sentences satisfying the rules as a summary. In addition to the frequencies of keywords and the locations of sentences, the proximities and distances between sentences are considered in determining the summary sentences. In result evaluations, the extracted summaries are compared with the sentences selected by human judges. The Chinese texts in evaluation are columns obtained from web newspapers because of their well-organized structures.

2.1 Introduction

With the growth of the internet, there are increasing number of documents in the electronic library. The free, direct and convenient source attracts users to access electronic documents. Although search engines can now effectively retrieve relevant documents, when a long list of relevant documents is found, it is impossible to read every document to determine if the documents are the desired items for a reader. Thus an abstract or a summary for documents is needed to provide a quick reference in an electronic text collection. However, not every document includes an abstract or summary and it is also time consuming to write a summary for each document. Hence automatic text summarization, which generates summary instantly when a new document is inserted into the text collection, can greatly improve the efficiency.

It is not an easy task for computer to generate a summary without natural language understanding. An author composing a summary needs a period of related training and instructions. In general, there are three steps to generate a summary. First, the whole passage is read once. Next all important phrases and sentences are selected. Finally the selected phrases are connected together with suitable connecting words to form a summary. This process involves human understanding and analysis of the document as well as grammar knowledge. It also needs selection of core sentences and phrases from the documents, discarding the trivial sentences and editing the final key sentences to generate a coherent summary. Thus it is hard to simulate this complicated summarization process and the human learning model for computers to produce a short and precise summary.

There have been many researches on the English summarization, whereas little have been done on the Chinese text summarization. When we are interested to find the local information, we may have to access the Chinese documents for more comprehensive information. Thus it is useful to built a system for summarizing Chinese documents.

Genetic algorithm is applied to find the optimal results in search space which is a *text* in the summarization system. Genetic algorithm is an adaptive search technique based on nature evolutionary selection. The solution is encoded into a set of parameters and represented as a *chromosome*, which contains *genes* for the parameters of a possible solution. Each combination of the parameters has its own unique *fitness* to measure the quality of the chromosome. The first generation of population of a group of chromosomes is generated randomly. To improve the quality of the population, the current generation reproduces the new population with *genetic operators*, including crossover, mutation and selection, which provide different ways to exchange genes between the old generation chromosomes. The parents are randomly selected according to the probability proportional to the fitness of the chromosome to perform reproduction. So the fittest strings can survive up to the last generation and have greater chance to reproduce children. The chromosomes are kept evolving from generation to generation until the termination criteria are met. The termination criteria can either be fixed as the maximum number of iterations or the fitness of the chromosomes has converged to a certain level. The fittest chromosome survived in the last iteration is the optimal solution in the search space.

The rest of this chapter is organized as follows. Section 2.2 discusses the related works in text summarization. Section 2.3 introduces the modeling of the genetic algorithm in the system. Section 2.4 explains the details of the Chinese text summarization process. Section 2.5 shows the experimental results. Section 2.6 and 2.7 discuss the limitations and draws a conclusion respectively.

2.2 Related Work

There have been many researches on summarization, most of which, however, focus on the English text summarization. The common approaches are word frequency, discourse (structure) analysis and machine learning. In using word frequency, the words appearing in the documents the most are considered the most important. The importance of the word is the weight of the word. The weight usually includes cue words or phrase (such as "in conclusion"), frequencies of keywords, the keywords' locations and their proximities. The total weights of the words in the sentences are then calculated and ranked. Since it is inaccurate to count word frequency only, variations and combinations with other methods are derived to obtain a summary with higher quality, although its computation time is short.

The approach of counting word frequency is commonly used in sentence selection and paragraph selection, which identify the sentences or paragraphs scoring the highest marks in ranking [1, 2, 3, 4]. The important sentences are then extracted and concatenated to form a summary. For sentence selection, concatenating the sentences selected without further analysis may lead to the mismatch of pronominal references, making the summary imprecise. For example, the key sentences containing the anaphoric references, such as "these", "those" and "that", and the pronoun, such as "he", "she" and "it" are extracted as summary while the sentences introducing the objects' names are not selected. Without including the source names, reader cannot know what exactly the objects are referred to. Thus the text should be traced backward from the selected sentences containing the anaphoric references and pronouns to obtain the source names.

A similar method to sentence selection is paragraph selection [2], in which a *text relationship map* is used to relate the paragraphs to each other. A text relationship map is a graph in which nodes are paragraphs and the links between nodes are to connect the paragraphs with threshold similarity. The paragraphs similarity is calculated by using IR method and is the number of overlapping terms between two paragraphs. The term weights depend on the number of terms in that documents and the collection of documents.

To select summary paragraphs from the document, the highly brushy node, which has many links connecting it to other nodes, has to be found. The paragraph represented by this node is a good overview paragraph and is suitable to be a part of summary because it has many overlapping term with other paragraphs. To provide more coherent extract, the depth-first path, segmented bushy path and augmented segmented bushy path are constructed. Depth-first path is built from the first node or a highly bushy node to the next most similar node. Segmented bushy path is to include the special topic which is represented as the nodes well connected to each other, but poorly connected to other nodes. Augmented segmented bushy path is to select the introductory paragraph from every segment.

Paragraph extraction has eliminated the problem of pronominal references and coherence in sentence extraction. Since a whole paragraph is extracted in paragraph selection, the integrity and coherence of the resulting summary is better than that generated by sentence extraction. However unimportant sentences are also included in the summary while the important sentences in other paragraphs cannot be extracted because the overall rankings of the paragraphs are not high. If the number of trivial sentences are large, this makes the summary not concise enough. In addition, the approach may not be appropriate for a short text with only several paragraphs. The length of the summary may still be too long compared with that of the original text, because at least one or two paragraphs are extracted. The resulting summary cannot be brief if the paragraphs in the text are long. The precision will be low in this case. Hence the performance of paragraph selection is suitable for summarizing long texts, such as books.

Another method to determine the important sentences is using within-sentence clustering techniques [5]. The summary sentences are those sentences having greatest cluster weight. A cluster is a sequence of consecutive words in a sentence the beginning and the end of which are keywords where the distance between the two keywords in the sequence of words should be fewer than a threshold. The cluster weight is the total weight of all significant words within a cluster divided by the total number of words included in the cluster. A sentence may contain more than one clusters. The maximum weight of clusters is the weight of the sentence. Apart from treating frequent words as keywords, frequent words will also be treated as important word when the words are not found in the dictionary which implies the words are rare. However the extraction does not emphasize "highlight" and "italic" words.

For such long passages as electronic books, Thematic Hierarchy Detection [6] can be used for text summarization by discourse analysis. The algorithm generates a about one-page summary to facilitate reader to understand the text content without browsing the whole book. It is based on the decomposition of text into thematic textual units of about one paragraph. The algorithm consists of three stages. In the first stage, it detects the thematic hierarchy of a source text to decompose the text into an appropriate number of textual units of approximately the same size. In the second stage, it adjusts each boundary between these textual units to identify a boundary sentence, indicating where a topic corresponding to a textual unit probably starts. A leading sentence that probably indicates the contents of subsequent parts in the same textual unit is then extracted. In the last stage, it generates a structured summary of these sentences, thereby providing an outline of the thematic hierarchy of the source text.

This one-page summary generation algorithm only extracts the sentences in the leading part of every topic. It does not address problems due to a lack of textual coherence and readability which always exist in sentence extraction algorithm. To improve readability, the algorithm divides every one-page summary into several parts, each of which consists of a heading-like sentence followed by some parameters.

Alternate approach applying genetic algorithm is used for extracting keyphrases. GenEx algorithm [7] is a hybrid of the Genitor steady-state genetic algorithm and the Extractor parameterized keyphrase extraction algorithm. The Extractor reads an input text and produces a keyphrase list. The parameters in the Extractor determine the processing of the input text. Then GenEx adjusts the parameters in Extractor to improve the extraction results.

Text summarization can be treated as a task of data mining. From the view of data mining, the text is the raw data while the summary is the undiscovered knowledge in the data. The approach of data mining of keyphrase extraction has been implemented in the past. Unsupervised learning was used to find two-word keyphrase [8] with Adaptive Resonance Theory (ART) neural network. But this approach produce a long list of phrases with low precision and the number of words of keyphrase cannot be varied. Baynesian Approach [9] has also been used to implement a keyphrase extraction as a supervised learning problem.

2.3 Genetic Algorithm Approach

In this summarization method a group of sentences are searched and extracted as the summary. From the analysis of articles and summaries written manually [10], the summary length is almost independent of the document length and it is around 85-90 words, or about five sentences. Although English text corpus is used in the statistical analysis, similar situation can be considered in Chinese summary. For a text with number of words less than a threshold, say 1500, 20% of the total number of sentences in the text are extracted as summary. But in some Chinese texts, the sentences are so long that some sentences consist of several clauses separated by many commas. If the number of words in a document exceeds an upper limit, say 1500 characters defined as a parameter in the system, a maximum of seven sentences will be selected.

For a long document, there are many combinations of the sentences to form a summary. In this case, genetic algorithm, which is a greedy search to find the near optimal solution in the search space, is better than exhaustive search to reduce searching time. With the use of genetic algorithm, the relationships between the extracted sentences can still be considered. Thus the quality of sentences extracted will not be lower than that of sentences the extracted from exhaustive search while the searching time of the genetic algorithm is lower than that of exhaustive search.

In the system, the definition of a sentence is a group of words ending with a full stop(" 。 "), a question mark(" ? ") or an exclamation mark(" ! "). In other words a sentence is the smallest group of words with complete structure and can be easily identified by finding the ending punctuation marks. Since the structure of a sentence is complete, it is used as an extraction unit for summarization in the system to minimize the needs to edit the final resulting summary. All it is required to do after selecting sentences is to change the anaphoric references in the concatenated summary to proper nouns. Although keyphrase extraction can make the summary more precise and short, the process of combining the phrases together involves grammar analysis and natural language processing. It is not an easy task and is out of scope of this work. Thus sentence selection is used in the system for its simple implementation while giving satisfactory results.

The genes in a chromosome are the sentence numbers. In preprocessing, each sentence is given a unique identification number as *sentence id.* Since a chromosome indicates the sentences to be extracted as summary, the genes in a chromosome cannot be duplicated. The length of the chromosome is the number of sentences in summary, which is calculated as 20% of total number of sentences in the short original text. So if there are 15 sentences in a document, the summary length will be $15 \times 20\% = 3$ sentences. A chromosome of another example of a text with 35 sentences is shown as Figure 2.1. The genes representing a sentence id in Figure 2.1 are 0, 5, 8, 15, 30 and 34. Each gene is unique within the chromosome and it is not allowed to have a chromosome with repeated genes like 0, 5, 8, 15, 30 and 30. The genes are sorted in ascending order to facilitate implementation.

0	5	8	15	30	34

Figure 2.1: A sample chromosome with six genes which are the sentence ids.

2.3.1 Fitness Function

A fitness function is formulated to evaluate the quality of the summary formed by the group of sentences of a chromosome. The fitness function is based on the number of keywords within the sentences, the lengths of sentences, the positions of sentences, the distances and the similarities between sentences. A set of heuristic rules is applied to calculate the fitness of chromosome. The fitness function expressed in mathematical form is as follows:

$$\begin{aligned} Fitness &= \sum_{i=0}^{n} \Big(\frac{(\text{\# of keywords} + \text{\# of proximities})^2}{\text{total words}} \\ &\quad + w_1 F(x_i) + w_2 G(x_i) + w_3 H(x_i) - w_4 Sim(x_i, x_j) \Big) \end{aligned} \tag{2.1}$$

where w_1, w_2, w_3 and w_4 are the weights, and

$$F(x) = \begin{cases} 1 & \text{if } x \text{ is the sentence in the first paragraph} \\ 0 & \text{otherwise} \end{cases}$$

$$G(x) = \begin{cases} 1 & \text{if } x \text{ is the first/last sentence in the paragraph} \\ 0 & \text{otherwise} \end{cases}$$

$$H(x) = \begin{cases} 1 & \text{if } x \text{ is the sentence in the same paragraph} \\ 0 & \text{otherwise} \end{cases}$$

$$Sim(x, y) = \frac{\text{\# of words matched in } x}{\text{total \# of words in } x} \times \frac{\text{\# of words matched in } y}{\text{total \# of words in } y} \tag{2.2}$$

The score of the fitness function is proportional to the number of keywords in the sentences and inversely proportional to the length of sentences. Since according to a statistic 70% of the summaries contain the first sentence of the text [10], an additional score will be added if the sentence in the chromosome is the first sentence in the text or it is the first or last sentence in the paragraph. However, the fitness will be penalized if the sentences in chromosomes have similar words and proximities. Another case the fitness score will be reduced is when the distance between the sentences are close together, say in the same paragraph. Since one paragraph usually has one main point, the sentences in a paragraph have similar content. The explanation of the fitness function will also be mentioned in Section 2.4.

The number of keywords and proximities in sentences are the most significant factors in computing fitness, so the weights of other factors, w_1, w_2, and w_3, are less than w_4.

Figure 2.2 is the flowchart of the genetic algorithm in the system. At the beginning of the GA, the initial population is randomly created. An individual is randomly selected based on its fitness and performs one of the genetic operations according to the probabilities of reproduction, crossover and mutation. The fitness of the child produced is then evaluated. The process of the individual reproduction is repeated again until the number of children reaches the population size specified in the system. When the number of children is equal to the population size, a new generation is produced. The whole process is completed when the number of generations equals the given generation size.

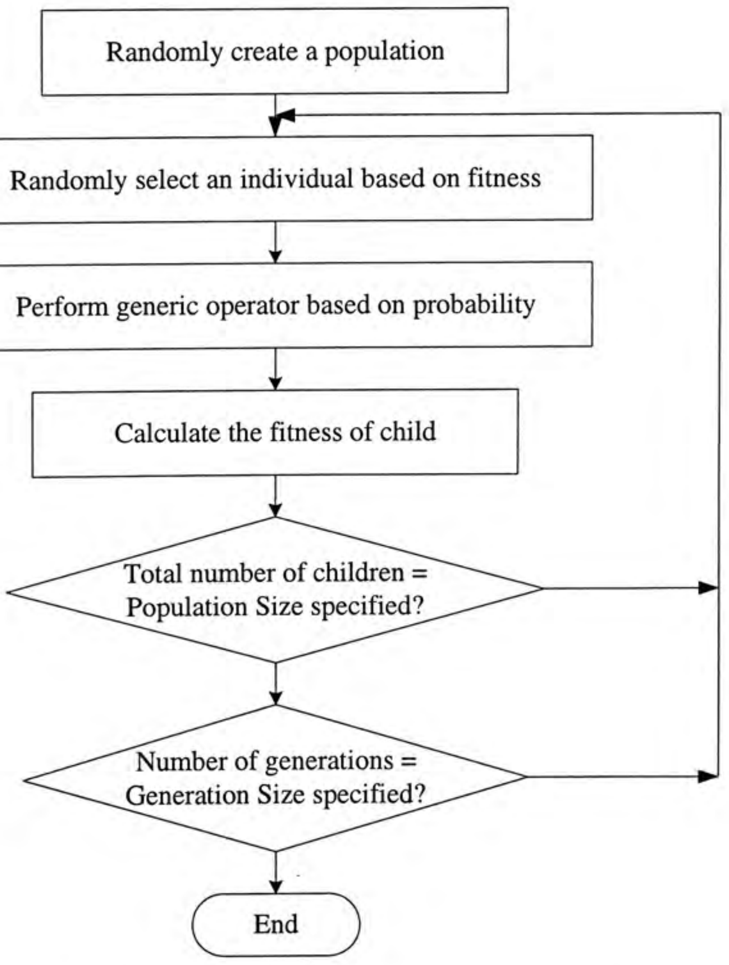

Randomly create a population
Randomly select an individual based on fitness
Perform generic operator based on probability
Calculate the fitness of child
Total number of children = Population Size specified?
Number of generations = Generation Size specified?
End

Figure 2.2: The flowchart of genetic algorithm of the system.

2.3.2 Genetic operators

Three genetic operators are used for reproduction in the system, which are crossover, mutation and selection. Crossover is to exchange the parts of genes between two chromosomes. The points where the chromosomes break and the number of break points of chromosomes are randomly determined. Since the numbers in chromosome should be unique, a new number has to be regenerated randomly if the numbers in chromosome are repeated after performing crossover. Figure 2.3 shows two chromosomes performing crossover. Because the two genes are repeated in the Child 1 after crossover, mutation is performed in one of the duplicated genes to generate a new unique gene to replace to the old one.

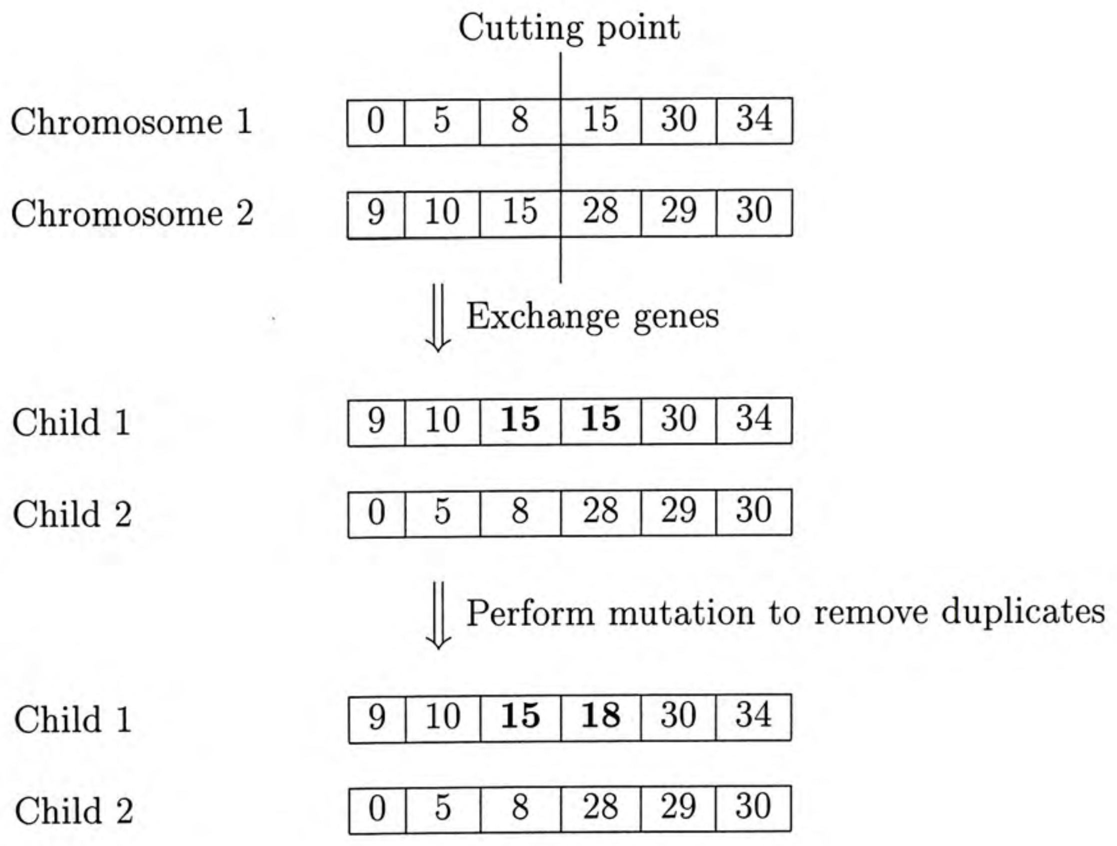


Figure 2.3: The process of crossover.

Mutation is another genetic operator which changes genes in a chromosome. The genes in a chromosome and the number of genes to be mutated are selected randomly. Similar to the crossover, if the genes in a chromosome are repeated after mutation, the genes have to be re-mutated until all the genes in the chromosome are different and the child is different from its parent. The mutation process is shown in Figure 2.4. In selecting chromosomes for performing reproduction, crossover and mutation, only the fittest 60% of the chromosomes in the population will be chosen for keeping the quality of the next generation above a certain level. The fitter the chromosome is, the higher probability it will be selected.

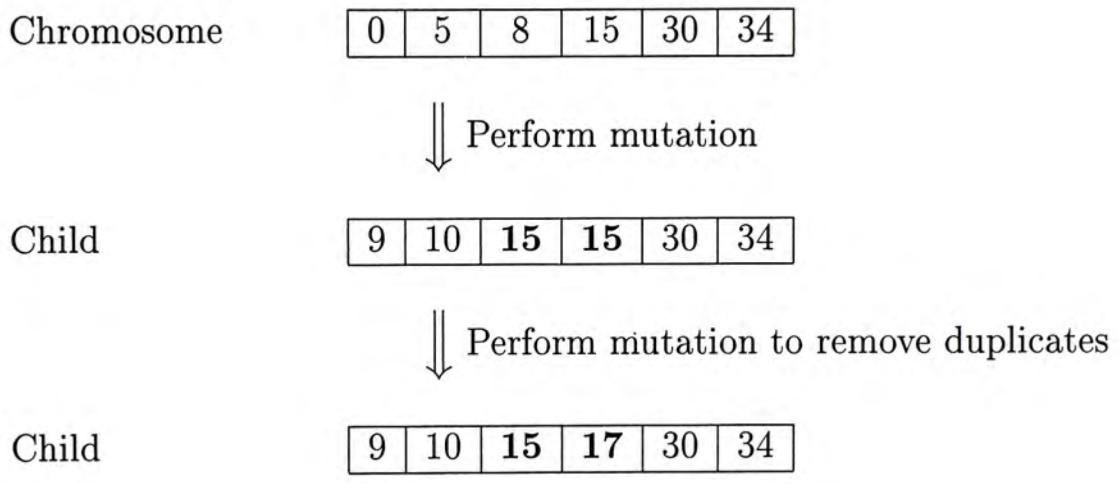


Figure 2.4: The process of mutation.

Elitism is also applied in the genetic algorithm. Elitism is to keep the chromosomes with highest fitness in the population to the next generation in order to prevent the best individual from being lost in the next generation and to prevent the highest fitness in the population from decreasing with generations. In our system, the best 1% of the population is kept.

2.4 Implementation Details

Before doing text summarization with the genetic algorithm, the text should be preprocessed. The system architecture is shown in Figure 2.5.

Since the input text is in Chinese and there are no spaces separating the Chinese words like that in English, this makes the analysis of the Chinese sentences difficult. A Chinese text segmentator is built to insert delimiters between words. For example,

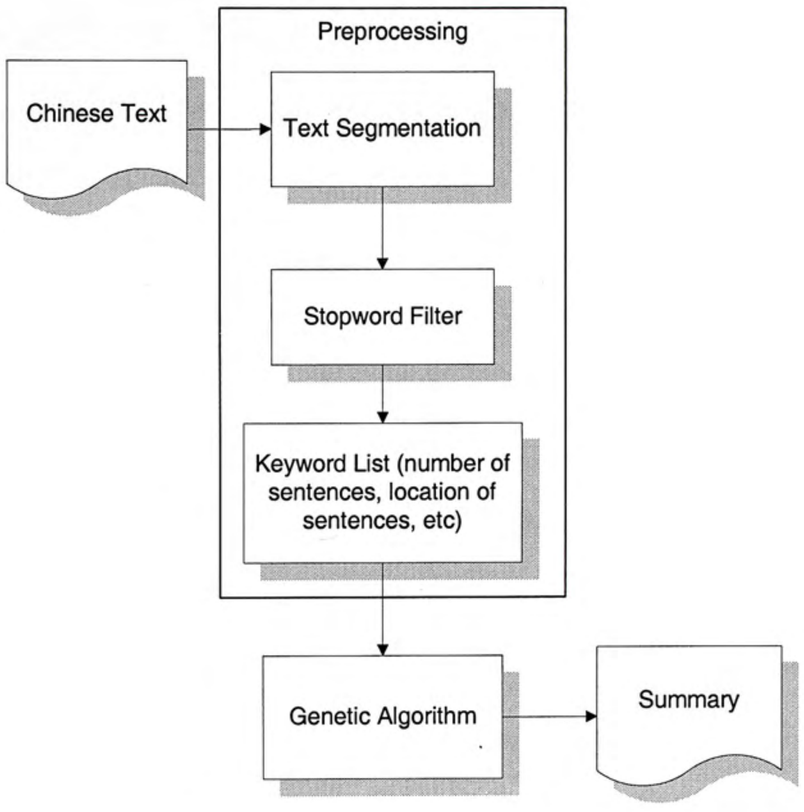


Figure 2.5: The system architecture of the Chinese text summarization.

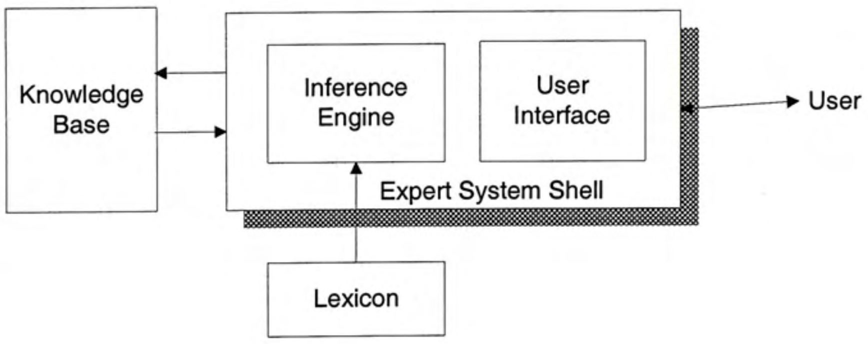


Figure 2.6: System Architecture of expert system.

```
if group = pure_numeral
and group = special_quantity
and word/group = pure_numeral
and word = special_quantity
and word/group = pure_numeral
```

Figure 2.7: A sample rule in word segmentation expert system.

the text segmentator inserts delimiters into the sentence "今天天氣很好" to produce a segmented sentence "今天 天氣 很好", where the phase "今天", "天氣" and "很好 " are separated by space. The Chinese text segmentator in our system is a rule-based expert system with the structure as shown in Figure 2.6. Forward Maximum Match is used to match the maximum length of words, from left to right, with the vocabularies in the dictionaries. A knowledge base contains all the segmentation rules for special cases so that the segmentator can segment the text generically according to the rules. A sample rule is shown in Figure 2.7. The rule specifies the condition the system should insert a space to separate the words. In the sample rule shown, it groups the words consisting of pure numeral, special quantity, pure numeral, special quantity and pure numeral in order together, such as "百分之一點五五", where "百", "一", "五五" are pure numerals while "分之" and "點" are special quantities. The number in the consequent part of the rule specifies the location the space should be inserted in. In the sample rule, the space is inserted after five objects in the antecedent and the segmentation result becomes "百分之一點五五 ". The segmentation expert system breaks down the sentences according to the rules in knowledge base. The rules will be checked only when no words are matched with the vocabularies in the dictionaries during Forward Maximum Match. Forward chaining is used in the system to find an appropriate rule in the knowledge base. In our system, each rule in the knowledge base will be checked in sequential order until a rule is satisfied. Thus only one rule will be fired.

It is not important even if the word segmentator cannot be 100% accurate, because the main aim of the text preprocessing is to identify keywords. The accuracy of the text segmentator will not affect the summary results greatly. In our system, those words missing in the dictionaries and cannot being processed with the rules in the knowledge base will be combined together and separated as single characters by delimiters. If words which should be grouped together but are mistakenly separated by delimiters because of its absence in the dictionary, the event extraction system would still work because the words in this case are not important words or special words. For example, the segmented sentence of "特區成立慶典在即" is "特區 成立 慶典 在即", where "特區" is not found in the dictionaries.

The accuracy of the segmentation module in our system is 97%. The main defect of the segmentation results is due to the absence of words in the dictionaries or unknown words, especially the idioms. Because there are an infinite number of Chinese words and idioms as well as those phases which are created by authors themselves, it is impossible to include all the words in the dictionaries. However this kind of segmentation error results in additional single words in counting the number of words, thus making the evaluation of the fitness of solution in GA imprecise. The original fitness function is *number of keywords / sentence length* instead of (*number of keywords* ^{2})/ *sentence length* as in the current version. The square of the number of keywords can amplify the effect of the existence of keywords, especially when there are many keywords in a sentence. For fairness, the fitness should be inversely proportional to the sentence length to ensure that the results do not show bias towards the short sentences and are more accurate.

In the next stage, the keywords are identified. All the stopwords are first filtered out from the segmented text. Stopwords are the words which appear frequently in texts and cannot differentiate the importance and uniqueness of sentence, such as 的, 我, 是. We mainly rely on the stopword lexicon to identify the stopwords. Similar to the problem in segmentation, some words cannot be identified as stopwords if the words do not exist in the lexicon. Since most of the stopwords are morpheme, which are single-character words, if the length of the words is one Chinese characters, the words will be treated as stopwords. This measure prevents the stopwords from being wrongly identified as keywords.

The frequency of other words are counted and the words exceeding the threshold frequency will be treated as keywords. To improve the accuracy, the proximity of the words is also considered in counting the words frequency. If the meanings of words are similar with each other, i.e. they are synonyms, their frequencies are summed up together to get the total frequency. All those words above the frequency threshold are the keywords. Other properties of the text, such as number of paragraph, total number of sentences, number of sentences in paragraphs and number of words in the sentences are obtained in this stage.

Finally GA makes use of the keywords and the useful information obtained from text preprocessing to generate a summary.

2.5 Experimental results

The control parameters of the genetic algorithm of our system are shown in Table 2.1. The system is written in C++ and the experiment was performed under Sun Ultra 5/270 with 256MB ram in Solaris 2.6 Platform.

Population Size	100
Probability of Crossover	0.5
Probability of Mutation	0.2
Percentage of Elitism	0.01
Number of GA iterations	100

Table 2.1: The control parameters of the GA.

Figure 2.8 shows the maximum fitness among the population of each generation of three texts in the experiment. **Text 1 and 2 are shown in the Appendix A while text 3 is shown in the Figure 2.9.** The characteristics of the three texts are listed in Table 2.2. Because the number of summary sentences of the text 1 is more than others, its fitness score is highest among three texts. As shown in Figure 2.8, the fitness of the population is increasing exponentially with generation. This shows the fitness of each generation is evolved and reaches the optimal results after several generations. In general, the more sentences the text contains, the more number of generations it needs to obtain the optimal results. Because the search space of the long text is larger, it needs more generations to evolve to get the optimal results. In Figure 2.8, text 1 finds the optimal solution after about 15 generations while text 2 and text 3 find the optimal solution in less than 10 generations.

	Text 1	Text 2	Text 3
Number of sentences	37	26	15
Number of words	660	463	541
Number of characters	1558	1126	922
Number of summary sentences	7	5	3
Execution time (sec)	127.69	54.36	39.35

Table 2.2: Main characteristics of three texts in Figure 2.8.

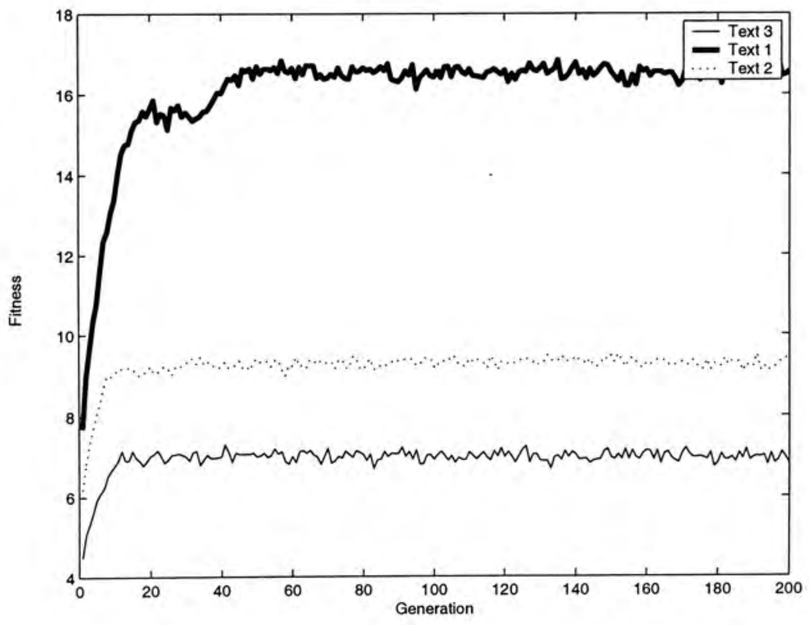


Figure 2.8: Fitness of the summary sentences of three texts over 100 generations.

Although genetic algorithm is a stochastic system, the summarization results generated each time are almost the same. The searched results of 9 out of 10 runs are the same optimal solution, while the remaining one is the sub-optimal solution. Table 2.2 also shows the execution time of each text. The execution time is counted by taking the average time of running the system for three times. The time can be much shorter if the number of generation is smaller because, as seen in Figure 2.8, the optimal solutions are already found well before 100 generations. So the number of generations can be adjusted according to the number of sentences in the text to improve the efficiency.

It is difficult to evaluate the summary results because it is subjective to determine which sentences have to be chosen for summary. To evaluate the results accurately and fairly, in an ideal case, the abstract written by the author can be compared with the generated summary. But there seems to be very few texts consisting of the abstract in the Chinese publications, so this method cannot be used.

To evaluate the system, an evaluation method similar to the "ideal" summary based on evaluation method [11] is applied. The summaries obtained by the genetic algorithm are evaluated with the "ideal" summaries created by a group of human subjects and the summaries generated by the Microsoft Summarizer (in Word2000).

The documents used in the experiment are columns obtained from the web newspapers. Columns are used instead of news in the evaluation, because it is easy to collect them from the web and the structure of it is suitable for summarization. It is found that the first paragraph of the news is already a summary. The structure of the news is that the more important the ideas are, the earlier they appear in the news article. So it is meaningless to extract summary from the news articles. We have five human subjects to select the summary sentences. The human subjects are graduate students in the Department of Computer Science and Engineering at the Chinese University of Hong Kong.

In this evaluation, 40 columns are collected mainly from three local newspapers, which are Oriental Daily News, Apple Daily and Ming Pao Daily. The average number of sentences is 18 and the average number of words is 567. A word means Chinese characters grouped together after text segmentation. The properties of the collection of columns for evaluation are shown in detail in Table 2.3. Each human subject produces a summary for each column. The five summaries generated by five human subjects are then compared with that generated by our system and Microsoft Summarizer. In this evaluation, the sentences voted by the majority, that is at least three out of five human subjects, are considered as the correct results.

As in the information retrieval, precision and recall are used to measure the quality of the summary. The distribution of the selection results among human subjects is

Max number of sentences	37
Min number of sentences	8
Average number of sentences	18
Min number of words	332
Max number of words	855
Average number of words	567
Min number of characters	720
Max number of characters	1558
Average number of characters	1021

Table 2.3: Statistic of properties of documents collected for evaluation.

measured by its deviation from the majority. The agreement deviation is defined as follows:

$$\begin{aligned} \text{Number of deviations} &= \text{\# of mismatch with the majority opinion} \\ \text{Total number of agreements} &= \text{\# of human subjects} \times \text{\# of summary sentences} \\ \text{Agreement deviation} &= \frac{\text{\# of deviations}}{\text{Total \# of agreements}} \times 100\% \end{aligned} \tag{2.3}$$

In the above formula, the majority opinion is at least three out of five subjects agreeing with the sentence to be summary. The agreement deviation among the five human subjects for 40 columns is shown in Table 2.4.

Average Deviation	Maximum Deviation	Minimum Deviation
22%	35%	12%

Table 2.4: Agreement deviation among five human objects for 40 columns.

The length of the texts affects the agreement greatly, because the more sentences there are, the more diverse the range of choices is. But the range of selection results is usually rather concentrated, because some sentences are obvious to be summary sentences, such as the last sentence of the text, which draws conclusions, and the first several sentences in the whole text which introduce the background.

Genetic Algorithm		Microsoft Summarizer	
Precision	Recall	Precision	Recall
68%	70%	34%	38%

Table 2.5: Evaluation results of two summarization systems.

The precision and recall results of summaries generated by GA and Microsoft Summarizer is shown in Table 2.5. Since the genetic algorithm is a stochastic system, the average precision and recall results are taken by running the system for three times. Because the Microsoft Summarizer counts the number of summary words in summarizing text, the length of the abstract is finely tuned each time in setting the percentage abstract length of the whole text to match the number of summary sentences of the corresponding text in our system and human subjects.

The evaluation results show that the summary generated by the genetic algorithm is better than that of Microsoft Summarizer. Because the extraction of sentence depends highly on the density of keywords and proximities in sentences in our system, if the important sentences have not satisfied the requirement, it will be disqualified as a summary sentence. Figure 2.9 shows one of the columns in evaluation. The first sentence in the text is selected instead of the second sentence by our system, it is because the length of the first sentence is shorter than the second sentence, the density of keywords in the first sentence is high.

Some authors may write a very long sentence containing several clauses and several commas. The sentence may be sometimes as long as about 200 Chinese characters. If long sentences are extracted as the summary sentences, it may make the summary too long and not compact enough. In such case, human may extract the key phases or clauses from the suitable sentences, recombining the points to form a summary. But it is difficult for computer to identify the complete phases and clauses and regenerate a complete sentence by combining the phases and clauses together, because one needs to solve the co-reference and anaphora problems and involves natural language understanding. Thus the sentence extraction is the preliminary step for generating good summary.

2.6 Limitations and Future Work

There is still a room for improvement in the quality of resulting summary. For the Chinese text, the sentences are always too long, the conditions of which is seldom appearing in English. A good sentence should has a suitable length, say 1 to 3 phrases or clauses. However in some news articles, it is common to find that sentences contains much more phrases and clauses. If the long sentences are selected, the summary will be long and is not precise enough. So sentence selection fails to obtain a good summary in this case. However, as mentioned before, identifying the suitable clauses and reconnect the clauses and keyphrase to form a summary require grammar rules and part of speech tagging, and even natural language understanding. This involves another topic of research.

Although some sentences are too long for summary sentences, sentence refinement can trim the sentence to make the summary more concise. Because some sentences are too long and not the whole sentences contains the main points, if the whole sentences are extracted for summary, the summary will not be concise enough. In this case the clusters of extracted sentences containing the main ideas have to be identified and the irrelevant parts are discarded. Then the clusters of the sentence have to be reinserted the particular words, such as connecting words, to make the summary coherent.

Sentence compression is required to combine several sentences with similar meaning into a single sentence to reduce repetition. For example, there are three sentences, "I have an apple.", "You have an apple." and "John has an apple.". They can be combined to form a single sentence as "You, John and I each has an apple.". But this technique also requires nature language understanding.

特區政府最新在醞釀一項不知所謂的政策，名為「博物館公司化」。 **康樂文化事務署轄下十三間公共博物館，準備以一兩間為試點，公開「招標」，由私營機構管理。**

特區政府誕生三年零九個月以來，不但在民生政策的制訂執行出現「差不多先生」的粗疏，在文化精神層次的戰略層面，更是一無所知，認知一片空白。博物館如何可以「公司化」交由商人投標經營？請這個愚庸政府睜大眼睛看一看世界：沒有一個國家的主要博物館，不是由政府管理，也沒有一個國家的主要博物館帳面不虧損。但博物館為甚麼即使虧損也要由政府經營？***因為博物館是民族的榮耀、國家的尊嚴、聲威的確立、旅遊的熱點，博物館本身即使收取門票，不能填補博物館本身的開支，但一座名博物館為城市國家的招牌，會帶來巨大的無形收益。***

世界博物館之王的大英博物館，不收門票，自由參觀，館內有埃及、希臘、遠東、非洲等世界文物珍藏，維修經營的成本是天文數字。但世界遊客慕名而來，大英博物館沒有收入，但博物館周圍的咖啡座、旅館、書店、餐廳，長年門庭若市；遊客遊大英博物館，也會遊附近的倫敦塔和杜莎夫人蠟像院，觀賞歌劇，品嘗佳餚美食，整個大倫敦的旅遊業為之生色生輝。巴黎的羅浮宮、紐約的現代藝術博物館、馬德里的帕拉多博物館，這些都是環球旅遊業的珍珠，展示的是人類文明的瑰寶，本來就屬於公民所有，而且是公民權益和福利不可分割的部分，政府由公民民主選出，政府來管理這批財產，天經地義。

特區的博物館，不論規模內容，與國際名都的名博物館相比，當然只屬小兒科，而且投資者不懂文化、不識藝術，把一座博物館投標來經營。只會在「廣東十八世紀文人繪畫展」的場地出售日本漫畫；在「香港百年藝術展」的廊館出售本港連環圖與刀劍玩具；舉辦「電影欣賞會」，亦定必以商業收入為重，加映四級成人電影。請問，這是不是所謂「文化委員會」的原意？這是不是所謂康樂文化事務署的目的？特區政府的高官，每值國際級的音樂大師來特區演出，由康樂文化事務署的官僚來巴結權貴，壟斷十多行座位，向高官送贈免費票，讓他們在場外的記者攝影機前炫耀華衣美服，在場內觀眾席間打瞌睡「釣魚」，藉以自高身價，裝扮「文化」排場。博物館公司化，就是繼「版權修訂法」之後，這個滿嘴文化而又毫無文化的庸俗政府的又一劣作醜行。

Figure 2.9: One of the columns in the evaluation. The bold sentences are the majority opinion of human subjects. The italic sentences are summary sentences chosen by our system, while the underline sentences are summary sentences chosen by Microsoft Summarizer in 20% length.

特區
政府
博物館
文化
經營
世界
國家

Figure 2.10: Keyword list of the text in Figure 2.9. The keywords are to identify the summary sentences.

2.7 Conclusion

A new application of genetic algorithm on text summarization and the new heuristic rules determining the fitness of chromosome are proposed. In this algorithm, the relationship between the sentences, such as the similarity and distance between two sentences, are also considered in determining the summary sentences. But the quality of the text greatly affects the results. Some writers produce very long sentences, which although is a key sentence, the long length make the sentence less likely to appear in the summary. To produce a good summary, sentences extraction is just the first step of the high quality text summarization. For future work, the It is better to keep the summary concise by sentence combination and refinement to generate a readable summary.

Chapter 3

Event Extraction from Chinese Financial News

Information extraction (IE) is the natural language text analysis to locate and extract the specified items in text. It transforms unstructured texts into a structured text database for further research. In IE, machine learning is generally used to provide an automatic results generation. However it requires considerable time to construct and is not suitable for medium IE system. This chapter describes a word-matching method to discover the desired items from the Chinese texts. It uses a set of relevant words to match the target items. To increase the accuracy, we have also assigned a set of irrelevant words to filter out the noise results. We applied the system on discovering events mentioned in the Chinese financial news articles, a task that can be treated as text classification. The extracted events then form an event database, which then can be used in data mining to obtain frequent episodes for stock movement analysis and stock prediction. From experiment, the results show that it is comparable to the other IE systems built with machine learning. The result is also published in [12].

3.1 Introduction

Information extraction (IE) is the natural language texts matching with the predefined patterns or templates, which, once matched, extracts the desired key items from the original text. The extracted items are stored into databases for subsequent processing, such as data mining, to discover the knowledge from databases.

Many researchers have applied machine learning methods on IE [13, 14, 15, 16]. They learn and generate rule bases from the texts and then extract information according to the rules obtained. Even though the machine learners intend only to shallow understand the text, because it is difficult and still impossible to fully understand the natural language text, the IE systems applied with machine learning are complicated and it is time-consuming to construct.

The IE learning system RAPIER [14] uses relational learning, such as inductive logic programming, to build rules. Given a text database and filled templates, it matches the pattern of words around the slot-filler and the filler itself. A string of words matching with the given pattern is then extracted as a rule. Finally all the rules obtained are then compressed and generalized to produce the best rules to insert into the rule base.

SRV [13] is another system using a top-down (general to specific) relational algorithm for information extraction. It is to find the best solution of continuous words in HTML to fill a slot in a template. SRV uses a set of *token-oriented* features to map token to discrete values and another token. The system then learns rules from positive and negative examples.

[17] presents a hierarchical classifiers in text categorization for classifying the documents into different categories, some of which are divided into additional hierarchies. It uses a kind of learning method to select vocabularies and tune parameters from the training documents. While building the categorizer, the levels of categories are also determined.

Although machine learning provides automatic operations from learning and generating rules to extracting target items directly from text, most of these learning methods are supervised learning which also involves manual preparation that labels the target results with each training text document. But for the smaller IE systems, system construction time is one of the main concern in deciding the implementation method. Considering the similar amount of human effort needed in labeling the documents for machine learning, we derived a simple method for small or medium IE system.

In this chapter, we present a method which matches a pattern of words to find the desired items in the Chinese text. We have applied the method on event extraction from the Chinese financial news. Event extraction can be viewed as text classification. In event extraction each news article, if an event is referred in it, is extracted at least one event whereas in text classification each document is assigned a suitable category. The events obtained from the continuous days of news articles are saved in an event database which, along with the databases containing the stocks of the listed companies and market index, are analyzed using data mining technique to discover the frequent episodes. The mined frequent episodes can then be used for stock prediction.

3.2 Method

3.2.1 Data Set Preparation

Given a collection of news articles, a small set of articles (5%) is selected randomly as a training sets, which are then browsed manually to determine the event features. First a group of target event types is chosen. The training articles are browsed and a set of words associated with each event type is selected as **positive words** to differentiate the event types. We have also identified a set of **negative words** to filter out the noise of unwanted events in the results. The positive and negative words will be described in detail in the following subsections.

3.2.2 Positive Word

Positive words are the words relevant to the event types and frequently appear in articles when the articles are mentioning the corresponding events. Each event type has its own set of positive words. The synonyms are also in the positive word list, where synonyms, in this paper, also include those words which are similar in concept.

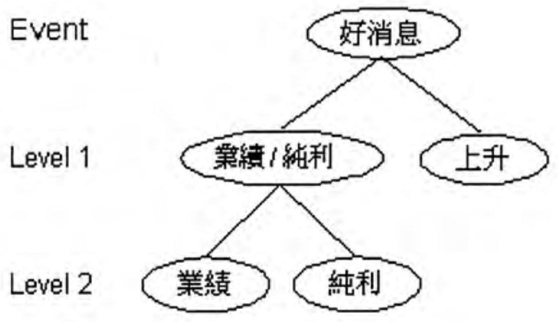


Figure 3.1: A tree diagram showing the relation of event, positive words and synonyms.

The relation between event, positive words and synonyms is shown in Figure 3.1. The tree shows the partial conditions for extracting the event "好消息" (good news from company). The tree functions like a decision tree. The tree root is an event for extraction and all words under the root node are the positive words for the event. The words at level 2 under the same child root node are the synonyms. When at least one of the synonyms exist, their parent node at level 1 will become *True*. Thus the condition of the node at level 1 to be *True* if it has synonyms is OR. For example, "業績"(company results) and "純利" (net profit) in Figure 3.1 are the synonyms, and their parent node "業績/純利" will be true if either "業績" or "純利" exist.

Level 1 consists of *all* the positive words required to exist such that the event can be extracted from the articles. In other words, the nodes at level 1 is under AND condition in order to make the root event *True*. For instance, in Figure 3.1 the word "上升" (rise) should also coexist *with* one of the synonyms "業績/純利" to extract the event.

When there are more than one groups of positive words for a single event type, another tree is formed. The tree in Figure 3.2 represents the other condition for extracting the same event that "特別股息"(special dividends) and "派發" (issue) must exist simultaneously.

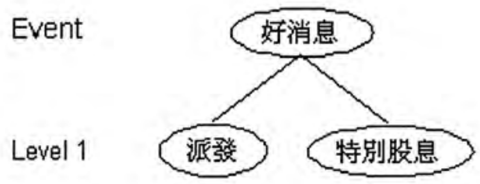


Figure 3.2: The second group of positive words for the event of "good news from company" .

Thus the overall condition combined two trees for extracting the event "好消息" (good news from company) is ((業績 or 純利) and 上升) or (特別股息 and 派發).

3.2.3 Negative Word

In some cases, even though all the positive words for an event are in the news article, certain words, known as *negative words*, once appear, the event associated is disqualified and will not be extracted. For instance, extracting events by matching only the positive words will also obtain those old events which have happened long time ago when the events are referred. Thus we have also chosen the words which appear with the re-mentioned events while do not appear with the fresh events. The negative words are mainly the words indicating the past events, such as "on Monday", "last week", etc.

3.2.4 Window

The target items we are going to extract are the events mentioned in the news articles. But the financial news articles usually only mention the events only in one or two clauses instead of throughout the whole articles. Thus we limit the length of words which should satisfy the conditions for extracting events to reduce the noise in the results.

The length of words is defined as a *window*. The actual window size in an article can be varied and depends on the type of document source. The unit of window size can be a word, a clause or a sentence.

With the use of window, the maximum distance among positive words and negative words is bounded by the window size, which increases the probability that the words are closely related that they are referring to the same event. And our aim in event extraction is to find the specific events associated with the corresponding bag of words. Thus the system does not need to recognize and check the orders of positive words and negative words in windows and the parts-of-speech of words.

3.2.5 Event Extraction

If there is no negative word within a window of words, the text in the window can determine the event types it mentions. In any tree representing the condition of extracting an event, when all positive words at level 1 are found and at least one word of each group of synonyms at level 2 is found, the event can be extracted from the news article. And more than one events from one article may be extracted.

The scheme of event extraction is independent of the frequencies of positive words that a positive word is counted once even if it appears more than once in a window. The scheme is different from that of topic categorization, in which the topic score depends on the frequencies of words. In topic categorization, the keywords are dispersed throughout the whole text, thus the topic is determined by the total frequency of all keywords for each category. However, in event extraction, the event is always mentioned in only one of the sentences in the first paragraph. The area of appearance of an event in the text is small, and we do not consider. The frequencies of positive words. In contrast, the coexistence of the positive words for an event is a significant factor in event extraction. Hence we treat all the duplicated positive words as appearing once when determining the event types.

3.3 System Overview

Our target is to build up an event database which stores the specific events happened within a certain period of time. The whole process is illustrated in Figure 3.3. The two other databases, which store (1) the stock prices of the selected listed companies in Hong Kong and (2) the background data, which is believed to have significant impact on the stocks of the listed companies, such as Hang Heng Index and Dow Jones Index, are also obtained directly from the open source. The three databases are analyzed with data mining to obtain frequent episode. The preprocessing method of preparing the event database, that is the steps in the gray rectangle in Figure 3.3, is discussed in Section 3.4, while the process of building up the stock database is described in Section 3.5. In Chapter 4, the data mining method of discovering the frequent episodes with the databases built will be introduced.

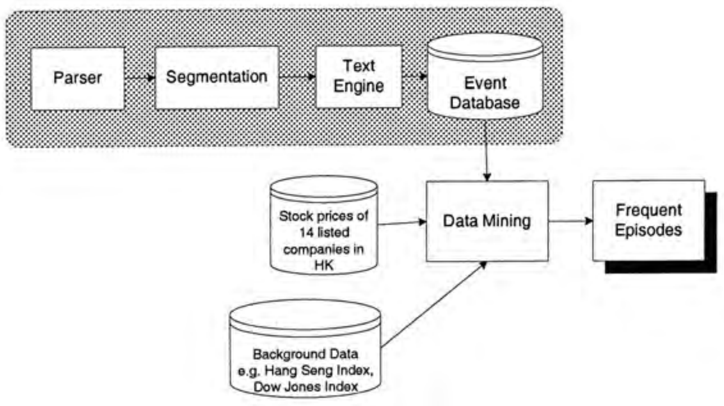

Parser
Segmentation
Text Engine
Event Database
Stock prices of 14 listed companies in HK
Data Mining
Frequent Episodes
Background Data e.g. Hang Seng Index, Dow Jones Index

Figure 3.3: System overview for the whole process.

3.4 Implementation

To obtain the events, we have built a web spider to collect financial news from four local Chinese newspapers, which are Apple Daily News, Ming Pao Daily, Sing Tao Daily and Hong Kong Daily News from 1 September 1999 to 29 September 2001. The HTML corpus is passed through a parser to skip the HTML tags. For Chinese documents, there are no word boundaries separating word from word like that in English. It is necessary to perform word segmentation to insert delimiters. We have built a segmentator that uses dictionaries and segmentation rules. The events associated are then extracted the Chinese news articles by matching words for each day of articles.

3.4.1 Event Type and Positive Word

From financial application viewpoints, we have identified 11 event types that are considered important. These events can appear in the local stock market, the government or overseas. Table 3.1 shows the 11 event types and the corresponding partial lists of positive words. The first four events are related to Hong Kong government and stock market while the events 5 and 6 are the US's decision in the change of the interest rate. The remaining events are the issues of the companies the stock of which we are interested. For each event, we have found a group of positive words by human judgment. One article may be classified into more than one event if it contains the positive words of the events and satisfies the conditions.

3.4.2 Company Name

Apart from the positive words, company names are to be identified for the events 7 to 11 in Table 3.1 which describes different issues of company. We have selected 14 main listed companies from various sectors, such as Banking, Property, Technology and Telecommunication. The stocks of these companies are active in Hong Kong stock market and their news are always reported in the articles. The companies selected are listed in Table 3.2. The abbreviations of the company names, such as "長實" (CK Holdings) and "新地" (SHK) of "長江實業" (Cheung Kong Holdings) and "新鴻基地產" (Sung Hung Kai Properties) respectively, are also identified.

Event No.	Event	Positive words
1	政府公布財政赤字 Government announces deficit	政府 財務 財政 赤字 Government, Finance, Financial, Deficit
2	發表施政報告 Report released	施政報告 特首 報告 Policy address, Chief Executive, Report
3	新基金上市 New fund released	基金 推出 面世 上市 Fund, Release, Debut, Listing
4	政府賣地 Government land auction	政府 成績 賣地 Government, Result, Land sell
5	美國不加息/維持利息不變 US interest rate unchanged	不加息 不變 利息 美國 Not increase interest rate, Constant, Interest rate, US
6	美國加息 US interest rate rises	加息 美國 interest rate rises, US
7	美國減息 US interest rate falls	減息 美國 Interest rate falls, US
8	收購 Takeover	收購 Takeover
9	新計劃 New plan from company	興建 簽約 合作 發展 計劃 投資 合併 Build, Sign contract, Corporate, Develop, Plan, Invest, Merge
10	壞消息 Bad news from company	裁員 業績下跌 純利下跌 Lay off, Results down, Profit down
11	好消息 Good news from company	派股 業績上升 純利上升 投資虧損減 Issue dividends, Results up, Profit up, Investment loss decrease

Table 3.1: 11 events and the corresponding partial list of positive words.

Stock Number	**Stock Name**
001	Cheung Kong Holdings 長江實業
002	CLP Holdings 中華電力
003	HK & China Gas 中華煤氣
005	HSBC Holdings 匯豐控股
006	HK Electric 港燈集團
008	HK Telecom 香港電訊
008	PCCW 電訊盈科
011	Hang Seng Bank 恆生銀行
013	Hutchison 和記黃埔
014	Henderson Land 希慎興業
016	SHK Properties 新鴻基地產
762	China Unicom 中國聯通
941	China Mobile / China Telecom 中國移動 / 中國電信
1186	Pacific Century CyberWorks Limited 盈科數碼動力

Table 3.2: Stocks the issues related to which will be extracted.

3.4.3 Negative Word

Since the news events may be mentioned not only on the day or on the following day the events has happened, but after a few days or a week. For example, weekly columns concluding the movements of stock market usually refer to the big news happened several days ago. If only the positive words are identified, the "re-mentioned" events will be misidentified as happening one day before the article's publication date and events with wrong dates will be produced. We have also selected a set of words that exist in the articles mentioning the old news but that do not exist in articles reporting the fresh news. These words are the negative words and part of which are listed in Table 3.3. The negative words are mainly the temporal words expressing the past or future events (e.g. Group 1 to 6 in Table 3.3), and the words showing expectation, fear or uncertainty(e.g. Group 7 to 10 in Table 3.3).

Group	Negative words
1	周一 (Monday), 周二 (Tuesday), 周三 (Wednesday)...
2	一月 (January), 二月 (February), 三月 (March) ...
3	上周 (Last week), 下周 (Next week), 本周 (This week)
4	上月 (Last month), 下月 (Next month), 月底(The end of month)
5	上年 (Last year), 下年 (Next year)
6	未來 (In the future), 前夕 (Eve), 即將 (Soon)
7	憧憬 (Hope), 或 (Probably)
8	預期 (Expect), 相信 (Believe) 料將 (Expect), 預料 (Expect)
9	對於 (For), 促 (Urge)
10	恐 (Fear), 陰影(Under the shadow of ...)

Table 3.3: Examples of negative words which must not exist within a window.

3.4.4 Event Extraction

In event extraction, every window size of words are scanned to find the event types. In implementation, we have set the window size to 1 to 3 clauses, where a clause is defined as a part of a sentence separated by punctuation marks, such as comma and full-stop. We use a 'clause' as a unit of window size instead of a 'sentence' because a clause is smaller than a sentence and gives more precise results. A sentence usually comprises multiple phases and clauses, and the rate of matching the unrelated positive words and company names will be increased if the window size is set to a sentence.

In scanning news articles, the current window is overlapped with the following window. For example, for the window size set to 3 clauses, the first window contains clause 1, 2 and 3, the second window contains clause 2, 3 and 4, and so forth, where clause x, $x \in [1, 4]$ represents the clause is the x-th clause in the news article, e.g. clause 1 is the first clause in the article.

When the following conditions are all satisfied in the same window, the events associated by the positive words will be extracted.

1. There is no negative word;

2. All positive words at level 1 of any tree, which represents the event and its

Date	Event
1/9/99	A
1/9/99	G8
1/9/99	J5
2/9/99	C
2/9/99	H6
⋮	⋮

Figure 3.4: The structure of the event database. The numbers following the characters in the second column represent the company ids.

associated words, are found and at least one word of each group of synonyms at level 2 is found;

3. If the event is a company event, the corresponding company name should also exist.

When an event is found, the date of the event, the event symbol and a company-id, if necessary, will be inserted into the event database. The event symbol is a unique symbol assigned to each event in Table 3.1 whereas the company-id is a unique integer assigned to each company in Table 3.2. Finally the extracted events are inserted into event database as shown in Figure 3.4.

3.5 Stock Database

In this section, we introduce the method of building up the stock database, which records the movements of the selected stocks in a certain period of time. First the stock movements are defined.

3.5.1 Stock Movements

The value of a stock at time t is denoted as $T(t)$, where t is measured in days. The average price of a stock is taken from the closing prices over the previous five days and is denoted as $Average(stock\_price)$. We define **up**, **down** and **flat** to describe the movements of stock price as follows:

A stock is defined as **up** if the percentage difference of the values of the stock at time t_1 and t_2 is greater than a threshold. For example $[T(t_2)-T(t_1)]/Average(stock\_price) \geq 1.5\%$, where $t_2 > t_1$.

A stock is defined as **down** if the percentage difference of the values of the stock at time t_1 and t_2 is lower than a threshold. For example $[T(t_1)-T(t_2)]/Average(stock\_price) \geq 1.5\%$, where $t_2 > t_1$.

A stock is defined as **flat, ie. neither up nor down**, if the absolute percentage difference of the value of the stock changes from time t_1 to t_2 is less than a threshold. For example, $|T(t_2) - T(t_1)| \ /Average(stock\_price) < 1.5\%$, where $t_2 > t_1$.

3.5.2 Implementation

In contrast to the event database obtained from newspapers, the historical financial data of stock prices are found from Datastream International Electronic Database. We have retrieved Dow Jones industrial average, Nasdaq Composite Index, Hang Seng Index Future, Hang Seng Index and prices of the 12 companies listed in Table 3.2 for the past three years. In addition, we have recorded the dates of Hong Kong public holidays.

3.5.3 Stock Database Transformation

The data collected from Datastream are the closing values of each trading day as in Figure 3.5. Since our goal is to find out what events results in the changes of prices, the numeric expression of stock prices cannot give us the clear idea of the movement of the stock. The stock price is therefore transformed to the symbols to indicate the

Date	**Stock Price**		
	Company A	....	Company M
1/9/99	62	....	85.9
2/9/99	62.5	....	87.0
3/9/99	64.67	....	85.1
:	:	:	:
:	:	:	:

Figure 3.5: The original stock database.

Date	**Stock Price**		
	Company A	....	Company M
1/9/99	-	....	-
2/9/99	Flat	....	Up
3/9/99	Down	....	Down
:	:	:	:
:	:	:	:

Figure 3.6: The transformed stock database.

overall movements of that day as shown in Figure 3.6. For example, the closing price of Company A on 2 Sept 1999 is transformed to 'Flat' since the difference of the prices on 1 Sept and 2 Sept is less than 1.5% and if the average price of Company stock A is taken as 61.5.

After building up all databases, the databases are analyzed to discover frequent patterns.

3.6 Performance Evaluation

3.6.1 Performance measures

The final discovered frequent episodes will be meaningful and useful if the given databases for mining are accurate. Thus it is necessary to make sure the events extracted from the newspapers are correct that they are same with the events extracted manually. To measure the accuracy of the event database, we evaluate the performance of event extraction using information retrieval (IR) performance metrics of precision, recall and F-measure defined as follows:

$$Precision = \frac{\text{Events found and correct}}{\text{Total events found}} \tag{3.1}$$

$$Recall = \frac{\text{Events found and correct}}{\text{Total events correct}} \tag{3.2}$$

where '*Events found and correct*' are the events found by the event extractor matching with that found manually for the same articles, '*Total events found*' is the total number of events found by events extractor and '*Total events correct*' is the total number of events found manually in the training set.

F-measure is the harmonic mean of precision and recall.

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.3}$$

where the precision and recall are given as equal weight.

3.6.2 Evaluation

We have randomly selected 600 news articles from the news collection, from which about 100 articles refer events. The 600 new articles are divided into training set (60%) and validation set (40%). The training set is the news articles we go through manually to obtain the positive words and negative words which then used to extract events from the validation set.

The financial news collection includes columns of critical essay and education for investors and other articles which may repeat mentioning events happened a number of days ago in the stock market. In order to filter out the undesired re-mentioned events from the final results, apart from using negative words, we limit the area in the news articles for event extractor to find the events. In the news reporting articles the main points are stated first whereas the details come behind. Thus the main events

Method No.	Area of news article searched
1	Headline
2	Headline with the first sentence
3	Headline with the first two sentences
4	Headline with the first three sentences
5	Whole article

Table 3.4: Five areas of news articles searched in the experiment.

we are going to extract are always located in the headlines and/or the first several sentences because the events we concern are important and have impact on the stock market. We have set five searching areas as shown in Table 3.4. Each searching area is assigned a number as listed in the first column of Table 3.4 for identification in the graphs of experimental results. The window and the words scanned will be within the searching area only. We have also set three window sizes: (1) 1 clause, (2) 2 clauses and (3) 3 clauses, to investigate the effect of window sizes on the accuracy. The performance of precision, recall and F-measure, for each searching area with each window size are evaluated and shown in Figure 3.7, Figure 3.8 and Figure 3.9 respectively. The method numbers on the x-axis of graphs represent the corresponding method shown in Table 3.4.

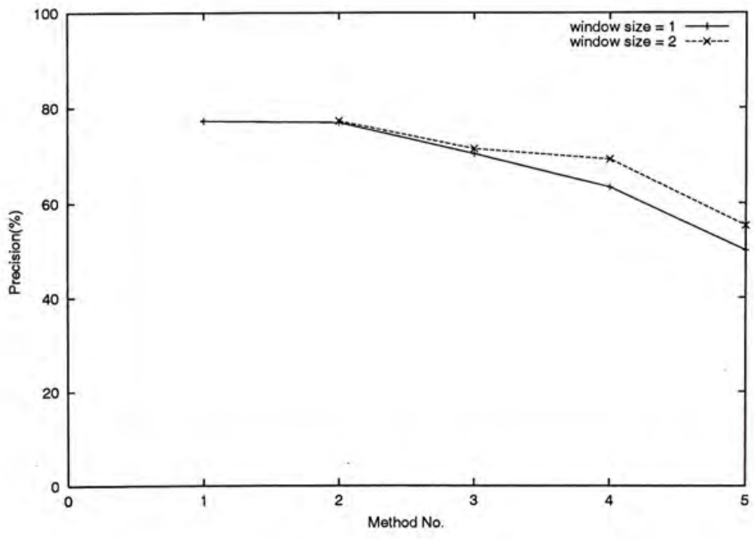


Figure 3.7: Precision with training set.

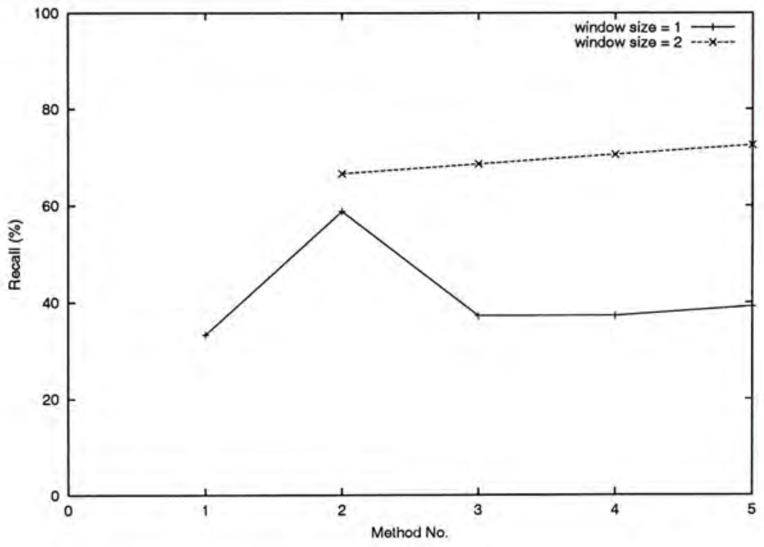


Figure 3.8: Recall with training set.

The performance of precision and F-measure in Figure 3.7 and Figure 3.9 gets lower with the increasing searching area in the texts. When larger area is covered in searching events from articles, the probability of the article extracted an event is increased. Thus '*total events found*' in precision is increased with the searching area. However the increasing number of events found does not mean the number of correct events found are also increased. It is because the structure of the news articles is that the headline and the first paragraph of the news-reporting articles state the key ideas, while the remaining parts of the article state the details and the trivial matters. Although searching the larger area gets the events that are missed when searching the smaller area, the increased searching area also implies increasing the chance of retrieving the incorrect events and reducing the accuracy. Incorrect events are usually extracted when the news article is not a news-reporting article but a column or an investors' education instead. Such articles always mention the events happening before and makes the system over-extract the events. However since there is no column name and no obvious keyword which can differentiate the types of financial news, the non-news-reporting articles cannot be filtered out. So in the case of searching the whole articles, the events extracted are always those events happened long time ago or those events the author expect/fear to happen. Thus '*Event found*

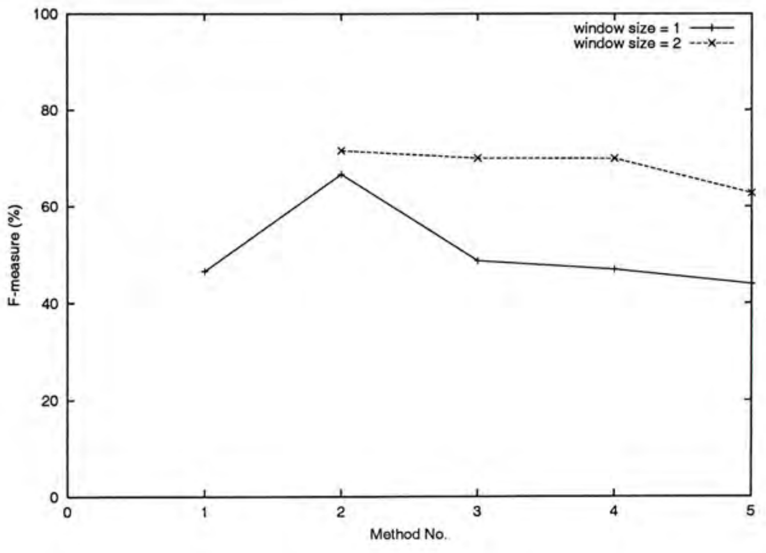


Figure 3.9: F-measure with training set.

and correct' in calculating precision is increased slowly relative to '*total events correct*' with increasing searching area, leading to the gradual fall of final precision.

In the performance of recall in Figure 3.8, we can clearly see how the searching area affects the number of correct events extracted. The shallow increasing curve shows the increasing searching area only slightly improves the performance. Comparing to the precision curve in Figure 3.7, the slope of the recall curve is obviously less steep. It shows the 'miss rate' of extracting event is greater than the 'hit rate' when the searching area is increased.

It is logical that the overall performance of window size set to 2 clauses is better than 1 clause and 3 clauses of window sizes. It is because some clauses are so short that they cannot contain all the positive words. When extracting the events concerning company issues, the company names and the associated positive words usually situate at different clauses. With a larger window size, more positive words and the company names can be included even if they are not in the same clause. But the window size should be bounded for limiting the maximum distance among the positive words and company names to ensure that all the words matched are closely related.

The performance of extracting individual events is also evaluated. Table 3.5 shows the performance in extracting the four main events: (1) US interest rate raise, (2)

Event Type	**Precision**	**Recall**	**F-Measure**
US interest rate raise	73.5%	69.4%	69.8%
US interest rate unchange	75.5%	70.5%	68.1%
Company releasing new plan	80.3%	75.2%	78.6%
Good news from company	81.3%	77.2%	79.4%

Table 3.5: The performance of the individual events.

US interest rate unchange, (3) company releasing new plans and (4) good news from company. The experiment is conducted with the searching area and window size set to "headline with the first sentence" and 2 clauses respectively. Among the four selected events, the first two events about US interest rate are the most popular events in the collection. But large proportion of the events referred are not the new events we desired. With the wide range of word expression in Chinese, it is difficult to cover all the possible expressions implying the undesired items. So the performance in extracting the event types of 'US interest rate' is lower than the other events.

3.7 Conclusion

We proposed a method to extract events from Chinese news articles. It uses word-matching method to recognize the associated events and extract the events if all the required conditions are satisfied. From experiment, we found that the performance is satisfactory.

Chapter 4

Mining Frequent Episodes

It is expected that stock prices can be affected by the local and overseas political and economic events. To understand better the relationship between the financial events and the actual stock market, the news are matched against stock prices databases for the mining of frequent temporal patterns from the stock markets and financial news events. We propose a new definition of frequent episodes which can better formulate our problem. This new definition does not have the property that the subset of a frequent episode must be frequent. We propose a new mining method for this problem which we show by experiment to be highly efficient and scalable. We applied our method on a real dataset collected from local financial news. We show that the method is effective in discovering interesting financial patterns.

4.1 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 fluctuation in the Hong Kong stock prices within a short 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-episode**, it can be written as "$\langle e_1, e_2, \ldots e_n$ (t days)$\rangle$", where the $e_1, e_2, \ldots e_n$ 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 $e_1, \ldots e_n$ appear in a window time of t days. Since we are only concerned with stock-episodes, we shall simply refer to stock-episodes as episodes.

4.1.1 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 an **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 occurs 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 [18], 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 which starts from day $i - x + 1$ to window 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 greater than or equal to a given **minimum support threshold**. $\square$

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

Note that the frequency of an episode is typically not equal to the number of windows that contain the episode, because if the same episode instance is contained in more than one windows, we only count the instance once. Suppose $D_1 = \{e\}$, $D_2 = \{a,b\}, D_3 = \{a,b\}, D_4 = \{f\}, D_5 = \{e\}$, and given a window size of 3, then there are two instances of the episode $\{a,b\}$, although 3 different windows contain the set $\{a,b\}$.

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 be the **window frequency** of a. The window frequency of a is typically greater than the frequency of a since the same occurrence of a is 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, the records D_1 to D_7 are: $\{b\}, \{a,c\}, \{b\}, \{d\}, \{b\}, \{c,a\}, \{d\}$, respectively. If the threshold is 3 and the window size is 3 days, 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. $\square$

4.2 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 [19], frequent "episodes" [18, 20, 21], temporal patterns [22] and frequent patterns [23, 24]. In [18, 20], 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 [25], but we would like to relate financial events with stock movement.

[19] provides the most classic Apriori-gen algorithm on mining the sequential patterns with point-based event. The main idea of Apriori is to find the k-length sequences from the (k-1)-length of sequences and each set of item of which must have the minimum support. A sequence is a list of set of items in the order of time. The algorithm first finds all the set of items in the database with minimum support. The set of items found become a set of 1-length large sequence and generates the candidate sequence with the length of 2. Candidate k-sequence is the concatenation of any two large (k-1) sequences with the same first (k-2) sets of items. Thus the candidate k-sequence contains the large (k-1) sequences. The candidate 2-sequence are counted with support in the database and those candidate sequences which exceed the minimum support are the large 2-sequences. The large-sequences with the length of 2, 3, 4 ... are continuously generated by the corresponding length of candidate sequences until no candidate sequences with minimum support are found. The resulting sequential pattern is the large sequences in all lengths with the minimum support and are not contained in any other sequences.

The Apriori approach improves the performance by removing the candidates the support of which is below the threshold. However it requires multiple scans of database and thus increases the cost when the database is large. It first scans the database to determine the set of items with minimum support. Then for each iteration of generating large sequences, the database is scanned again to count the support of candidates.

The Apriori algorithm is only suitable for finding the general patterns or behavior of a group of subjects. It is different when we want to discover the inter-relationship between events or sale transaction, the happening time and order of which should be taken into consideration. [18] 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. For example if a tax deduction and a company A stock rise occur in 2 consecutive days, and the window size is 4, then 3 different windows will contain the two events and the episode will be counted 3 times. However, there is actually only one instance of the episode, therefore we try to avoid this duplicated counting in our definition and in our algorithm.

[18] applies the main idea of Apriori [19] to find the k-length frequent episodes from the $(k-1)$-length frequent episodes. This is based on the property that any sub-episode of a frequent episode is also frequent, and that our definition does not guarantee. At the beginning, all episodes containing only one event are identified and the frequencies of the episodes are counted. The episode frequency is defined as the proportion of windows containing the episodes, where the window is the given width of the time in the event sequence. The episodes with the frequency threshold then become frequent episodes with one event. The single-event frequent episodes are then used to generate the two-event candidate episodes. Similar to Apriori, the frequencies of the two-event candidate episodes are counted and those episodes with

frequency threshold are the two-event frequent episodes. The process is reiterated to produce the frequent episodes with the larger numbers of events until the there are no candidate episode generated.

To generate candidate serial or parallel episodes of (k+1) events, the collection of frequent serial of parallel episodes of k events are sorted in lexicographically order. Each event type is denoted as a letter. If the episodes have the same first (k-1) events, they are located consecutively. The candidate episodes are the all combinations of the two parent episodes which are the consecutive episodes sharing the first (k-1) events. Thus a candidate episode is composed of the first (k-1) events of one of its parent episodes and the last events of its two parents. The candidate episode is then checked whether their subepisodes are frequent episodes. The candidate will be filtered out if their any of subepisodes are not frequent.

The frequencies of the episodes are counted after candidate episodes generation. To identify an episode, a time window is used to scan the event sequence before the start of the sequence and stops when the sequence is ended and the window is empty. When counting the number of serial or parallel episodes, the frequency of the episodes is increased by the number of windows the whole episode occurring in, when the number of events of the episode in the windows begins to reduce.

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. [22] discovers more different kinds of temporal pattern. In addition to the series and parallel events, the temporal patterns like "the duration of event A *overlaps* that of event B" and "event X and event Y start at the same time but X ends first and Y's finishing time *meets* the starting time of event Z". It used the approach similar to the Apriori-gen approach and the process is splited into two phases, which are "candidate generation" and "large k-item generation". The method starts with the large 1-item set in which the number of an event in the database is larger than the threshold. It generates k-candidate and large k-items from the large (k-1)-items and candidate k-items respectively. The k-items means the k

events with a particular temporal pattern in each pass. Thus "A overlaps B" and "A before B" are the two different 2-items. The generation of candidate and large items are reiterated until there are no large k-items found in the pass.

In the candidate generation, it also finds if the sub-pattern to be concatenated to form the k-candidate is the frequent pattern. That is, the item to be added should has the temporal relation with at least one event in the previous large items. It removes the redundant candidates to increase the efficiency. In generating large k-items, it first counts the support of each candidate. The end-time of large $(k-1)$-items and large 1-item are compared to decide if the temporal pattern is supported in the patterns from large $(k-1)$-items and large 1-item. The large k-items are produced from that sequences with minimum support.

[26] discovers frequent sequential patterns by using a tree structure. The sequential pattern it finds is the series of events in the order of time and can be represented as $A \rightarrow B \rightarrow C$, where A, B and C are events and come from the same domain, and which means A happens one day before B and C happens one day after B in which A, B and C last for one day. The algorithm applies a *trie*, a n-ary tree with n branches to find the frequent sequential patterns. A node in the trie represents an event. The events in the same level happen in the same day. And the events at a k-level node happen one day before the events at $(k+1)$-level. Thus the root event is the earliest event in the trie. For each event type, a trie with that event type as root is built. The events are attached to level 1 of tries if the events happen one day after the root event. After building a 1-depth tries, the support of each 2-event pattern in every trie, that is a path in trie consisting of 2 events, is counted that the fraction of the number of occurrences of the patterns to the total number of days in the database. For those patterns the support of which are less than threshold, they are pruned. The tries continuous to grow to more levels by attaching events nodes the patterns of which should appear in the database and by pruning the patterns less than support. The tries stop to grow when the event patterns in tries fully represent the events in database or the tries meet a day of breaks of events, that is the day nothing happens

or all the events in that day are already contained in the tries. The frequent sequential patterns are the deepest paths and all the sub-paths from root to all levels in all the resulting tries.

There are many limitations in the algorithm. [26] considers a break of events in the sequence of days as a break of sequential pattern. In many applications, the events before and after the break may be associated, hence some interesting sequential patterns may not be found. The algorithm also fails to find the more complicated sequential patterns. It restricts the events in sequential patterns which should occur one by one and each event should last for one day only. Thus it cannot find the patterns in which some events may be lasting for more than one days or in which an event A happens two days after an event B. The cost of the algorithm is quite high. For each phase of attaching a new level of branches to the existing tries, the database should be scanned once to find what the following events should attach to. The number of times of scanning the database is determined by the longest number of days of events without break.

Most of the algorithms introduced in the above are based on Apriori Algorithm [19]. However the Apriori approach scans the databases repeatedly and the cost increases considerably with the size of the databases. [23] provides a fast alternative to find the frequent pattern with a frequent pattern tree (FP-tree), which is a kind of prefix tree. The FP-tree is applied in transaction database, in which each transaction contains a set of items. A node in the FP-tree is a frequent item with the number of occurrences in the database no less than the support. A path of tree represents one or more than one transactions. FP-tree takes advantage of the structure of prefix tree that makes the tree compact by keeping the most frequent items near the top of the tree so that the frequent item nodes can be shared with a number of transactions. Thus the depth the item is located in is usually inversely proportional to its frequency in the database. Each node of FP-tree consists of*count* which records the total number of transactions containing the corresponding item in all the path(s) linking the node. For each item type in the tree, there exists a pointer, called *head of node-link*, in a *header table* to link all its same item types in the tree together to speed up searching in discovering sequential patterns. The items in the header table are arranged according to the item frequency in database. Thus the most frequent item is located at the top of the header table.

To build the FP-tree, it first scans the database. The frequency of each item is calculated to determine the number of transactions each item type appears in the database. The item types are then sorted in descending order of their frequencies and are formed *a list of frequent items.* The database is then scanned again. For each transaction in database, the items are sorted according to the order in the list of frequent items. The "*null*" root of FP-tree is created. Next the first item of the item-sorted transaction is compared to the FP-tree's child nodes. If the root of FP-tree has a child of that item, the count of that node will be increased by one. When the child node of the item is not found, a new child is created and its count is initialized to one. The remaining items in the transaction will be attached to that new child in the decreasing order of frequency with the most frequent item attaching first and the least frequent last. For the case of the first item in the transaction shares the existing child node of root, the second item of the transaction is then checked if it exists in the child nodes of the first item node in the FP-tree. The count of the child node of second item will be incremented by one if the node exists. Similarly, if the second item is not included in the child nodes of the first item in FP-tree, a child node with the second item type is created and its count is set to one. The new node and nodes of the remaining items in the transaction concatenate the prefix node in the descending order of frequency. The process is repeated for the remaining items in the transaction and for each transaction in database.

After building up the FP-tree, frequent patterns can be obtained from the tree. First all transactions containing a specified frequent item are found. By starting from the end of the header table for an item a with the minimum frequency, the link of that item header is traced and all the paths its node connecting in the FP-tree are found. The paths are then derived to *prefix path* of that item. Prefix path is the path before the item *a*. For example, for the path of <*f*:3, *p*:3, *m*:2, *a*:2>, the prefix path of *a* is <*f*:2, *p*:2, *m*:2>, where the counts in the prefix path are the same with that of item *a*, because the count are based on the frequency of the item appearing with item *a*. A FP-tree is then generated by all the prefix paths and the tree is called a *conditional FP-tree* of "*a*". The frequent patterns are all the combinations of the items in the path for a single path conditional FP-tree and can be with different number of items.

From experiments, it is shown that FP-tree involves a much shorter execution time than its previous works. It is efficient in mining frequent patterns, and it needs only two scans of the database. The first scan is to count the frequencies of items whereas the second scan is to build the paths in FP-tree. Since the access of the database is the major performance cost, the reduction of scanning the database can greatly decreases the running time of the algorithm. The compact structure of the FP-tree makes the tree more possible to be placed in memory instead of in disk thus further increasing the efficiency. 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 [27], but the problem is quite different from ours. In fact, our definition of frequent episodes does not give rise to the subset closure property utilized in many previous methods. Therefore it is not obvious how the problem can be solved efficiently. In this chapter we show that it is possible to apply some of the successful ideas in the FP-tree and derive an efficient mechanism for the mining of frequent episodes.

4.3 Double-Part Event Tree for the database

The method we proposed to mine the frequent episode has some similarity to that in [23]. 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. Before describing the details of the two phases, the tree structure will be introduced first. Two different tree structures are designed and the two trees have their own advantages either in implementation or performance. Except the tree structure, the overall mining processes with two trees are similar.

We propose a tree structure for storing the event database which we call the **double-part event tree** (DE-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 DE-tree is built, we first gather the window frequencies and the frequencies of each event type in the database DB. We sort the event types by descending frequencies and identify the frequent event types when their window frequencies are no less than the support threshold. 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 DE-tree. One tree node corresponds to each event type in each of F and 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.

The levels of the event types depend on the frequencies in database because the DE-tree is a prefix tree in which the levels of event nodes are determined from the frequencies in paths. Since the paths of tree are divided into two parts, where the first part contains the event types in the first day of window, the levels of the event types follow the descending order of the frequencies in the first part of the path, which is also the event types' frequencies in database.

On the other hand, the window frequency is to identify the frequent events for constructing tree. Since when we choose the event types to be included in the tree, the event types are likely to be members of the frequent episodes. So as long as the event types appear in windows, the event types can be parts of the frequent episodes. For the details of identifying the frequent episodes, please refer to Section 4.4.

In the DE-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 DE-tree differs from the 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 nearer to the root node form the first part, this part stores the event types in the first days of windows that are entered into the path. This is called the **firstdays part** of the path. The second part of the path stores the event types in the remaining days of windows inserted into the path. This is called the **remainingdays part** of the path. For the nodes in the firstdays part the binary bit is 0, for the remainingdays part the binary bit is 1, hence the binary bit indicates the location of the event type in the window.

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 DE-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 DE-tree structure is that windows with common frequent event types can likely share the same prefix nodes in the DE-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 a 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,

To facilitate our counting of episode frequencies, we have the following strategy:

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 4.1(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 $< b, a, c, d, x >$. The events in the two parts of window will contain these frequent events only and be sorted according to this order.

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

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	$< (x), (\phi) >$

(b)

Figure 4.1: An event database and the corresponding windows.

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 $< (b : 1, c : 1, a : 1), (d : 1, x : 1) >$, 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. A new header table is created in which the event types sorted in descending order of frequencies $< b, a, c, d, x >$. Each node is connected to the corresponding entry in the header table. The tree inserted the path of the first window is shown in Figure 4.2. The event types in each window are shown in Figure 4.1(b). The first column of the Figure 4.1(b) represents the order of reading the window. The second column shows the days covered by the corresponding window. The last column shows the sorted event types in windows after removing the unfrequent event types and duplicate events.

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 $< (b, d, x), (a, c) >$. Since the tree root has a child node of event type b, the path of the second window can share the existing b node. The count of the b node is incremented by one. The new nodes are linked to the existing nodes

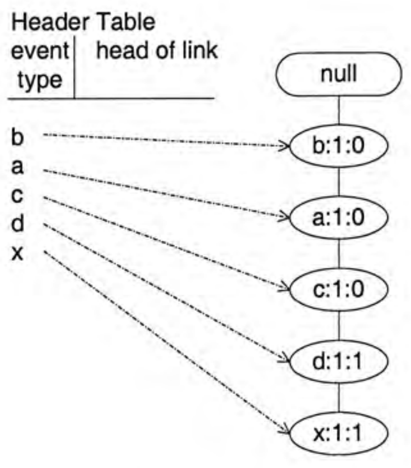
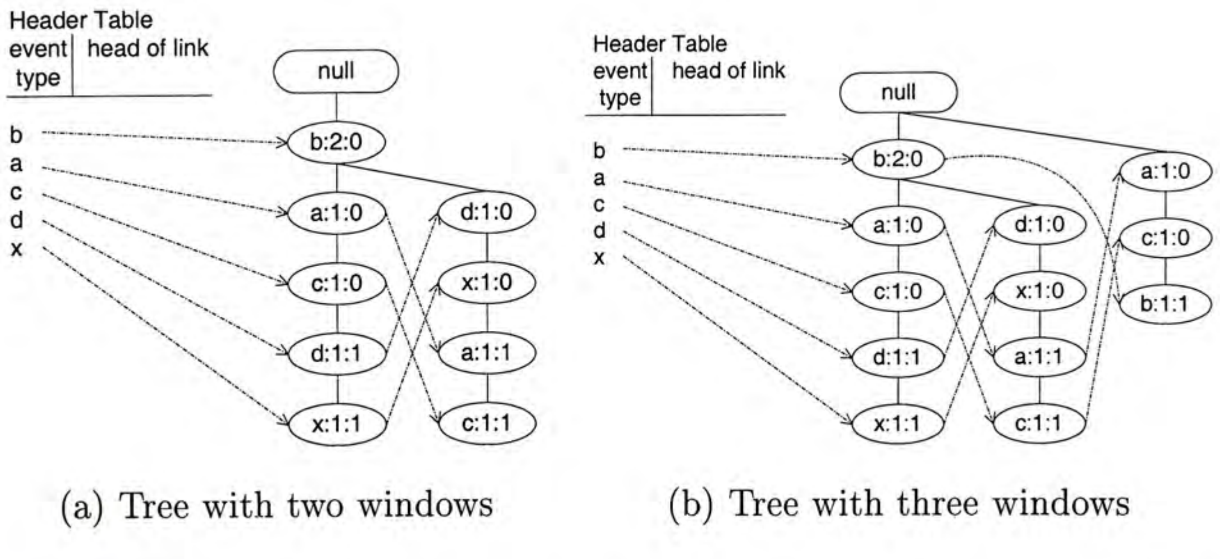


Figure 4.2: The tree after being inserted the first window $< (b, a, c), (d, x) >$. The content of nodes are represented as "event name:count:type".

with the same event types. The tree after inserting the path for the second window is shown in Figure 4.3(a). 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) Tree with two windows (b) Tree with three windows

Figure 4.3: The trees after inserting the first two windows and the third window $< (a, c), (b) >$ respectively.

The events in the fourth day of database are read, selected and sorted to obtain the third window, $< (a, c), (b) >$. A new path $< (a : 1, c : 1), (b : 1) >$ is created as shown in Figure 4.3(b) for the whole window because there is no child node a under the null root node.

The remaining windows are inserted to the tree in the similar way. The rough structure of the final tree constructed is shown in Figure 4.4. (Note that some dotted lines are missing in the figure for clarity.)

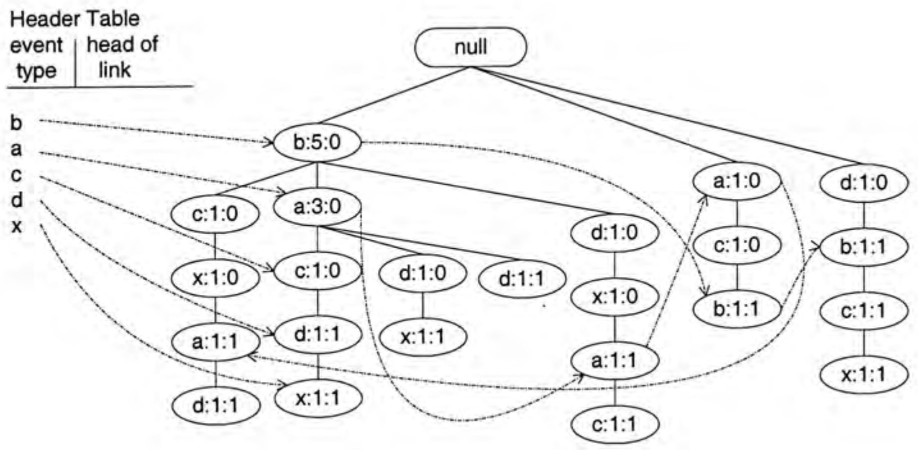


Figure 4.4: The final tree constructed in Example 1.

4.3.1 Complexity of tree construction

The DE-tree construction method scans the database twice in the whole process. The first scan is to accumulate the total frequencies in the database and in the windows, finding the frequent event types. The second scan is to read the window information to build or update the paths in the tree. The cost of constructing the tree is O(nm), where n is the number of windows which is also equal to (*number of days recorded in database*), and m is the maximum number of event types in a window. The cost of inserting a path is O(p), where p is the number of frequent events in a window.

The size of the DE-tree is bounded by the size of the database, but is typically much smaller because of sharing of frequent prefix sub-paths. In the worst-case, each window is represented by a unique path in tree. The number of paths is bounded by the number of windows in the database. The height of tree is determined by the number of frequent event types in windows. The longest path corresponds to the greatest number of frequent event types in a window.

4.4 Mining Frequent Episodes with the DE-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 DE-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 DE-tree T' with its header table in a similar way as the first DE-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 4.4, 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 4.4, 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.

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.

We shall now explain the four steps above for the path processing. The reason for Step (1) is quite obvious. We shall remove h_i (Step 4) in the path since we want to form a conditional database with all episodes implicitly containing h_i. For Step (2), we want the resulting path to reflect the number of occurrences of the event types together with h_i. Next we explain Step (3) in the above, where we move all events in the remainingdays part to the firstdays part. This can help to achieve both Objective B1 and B2.

For Objective B1, we need to find frequent episodes with h_i plus one other element from the paths. The reason for step (3) is that h_i will be removed from the paths, however, implicitly it is included in all episodes to be discovered. For a window W to be counted once for a set $b \cup h_i$, W must contain $b \cup h_i$ and at least one element in $b \cup h_i$ should appear in the first day of W. Therefore either (1) b appears in the first day, or (2) h_i appears in the first day. If (2) is true, we move b to the first day, then after h_i is removed in Step (4), we can simply use the count of b's in the first days of windows to count the occurrences of $b \cup h_i$.

Step (3) helps in Objective B2 as follows: suppose window W contains h_i in p in the first day, and the remaining days of W contains event types E_W. Therefore h_i combined with elements in E_W can be counted for episodes containing h_i. Since we shall remove h_i, when we later consider W, we may not count W for those combinations of h_i with elements in E_W since such no event type in h_i or E_W will appear in the first day. If we move E_W to the firstday part, we can be sure that these will be counted as well.

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.

4.4.1 Conditional Event Trees

In the previous subsection, we describe how we can form a conditional database DB' with a base event set $\alpha = \{h_i\}$, and a conditional DE-tree T' can be built from DB'. In the header table for T', the event types are sorted in descending order of the window 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 DE-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 DE-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.

4.4.2 Single Path Conditional Event Tree

Strategy 3 When a conditional DE-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 is a possible frequent episode. The union of any element of S_1 and any element of S_2 is also a possible frequent episode. And the frequency of such a episode is the minimum among the event types in episode.

Rationale By the way we construct a conditional DE-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 occur in the in the firstdays part. Therefore when we form an episode with an element in S_2 we must combine with some elements in S_1.

4.4.3 Complexity of Mining Frequent Episodes with DE-Tree

The complexity of the mining frequent episode depends on the number of paths and the depth of tree. It is NP-hard because mining association rules is a NP-hard problem. When the window size is 1, it is a special case that the problem becomes mining frequent patterns with FP-tree where the pattern corresponds to an episodes whose all events appear in the same day. The DE-tree has the same structure with the FP-tree except that the node types of all event nodes are 0 and are trivial.

4.4.4 An Example

Example 2: We illustrate the mining process with the event database in Example 1. Let us call the header table in Figure 4.4 HT. Starting from the event type x at the bottom of HT, we examine all the paths that contain x by following the links of the event type from the header table.

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. For the bottom element of HT, x, six paths are visited: $<(d : 1), (b : 1, c : 1, x : 1)>$, $<(b : 5, c : 1, x : 1), (a : 1, d : 1)>$, $<(b : 5, a : 3, c : 1), (d : 1, x : 1)>$, $<(b : 5, a : 3, d : 1), (x : 1)>$, $<(b : 5, d : 1, x : 1), (a : 1, c : 1)>$ and $<(x : 1), ()>$. Since the counts of some events are not the actual event frequency in the path, the count have to be adjusted before counting the total frequency of each event type in the event "x" path. The "real" count is the count of the event type "x" because the event type must appear with event type x simultaneously. Thus the paths become $<(d : 1), (b : 1, c : 1, x : 1)>$, $<(b : 1, c : 1, x : 1), (a : 1, d : 1)>$, $<(b : 1, a : 1, c : 1), (d : 1, x : 1)>$, $<(b : 1, a : 1, d : 1), (x : 1)>$ and $<(b : 1, d : 1, x : 1), (a : 1, c : 1)>$.

Conditional paths and conditional DE-tree: Next we can form the conditional paths for x. if the firstdays part of a path contains event type x, all the event types in the remainingdays part can be moved to the first part. Then the event type x in the paths can be eliminated. The conditional paths for x are $<(d : 1), (b : 1, c : 1)>$, $<(b : 1, c : 1, a : 1, d : 1), (\phi)>$, $<(b : 1, a : 1, c : 1), (d : 1)>$, $<(b : 1, a : 1, d : 1), (\phi)>$, $<(b : 1, d : 1, a : 1, c : 1), (\phi)>$.

Accumulating the counts of each event type in the firstdays parts and the remainingdays part of all paths, we obtain the window frequencies and the first-part frequencies, which are $<b : 4, d : 4, a : 3, c : 3>$ and $<b : 5, d : 5, a : 3, c : 4>$. Thus the frequencies of 2-event episodes containing x are $<bx : 4, dx : 4, ax : 3, cx : 3>$, which are not frequent. Since the window frequencies of event types a and c are less than threshold 5, we need not consider the episode containing xa and xc. We construct x's conditional tree with only two event types b, d as shown in Figure 4.5.

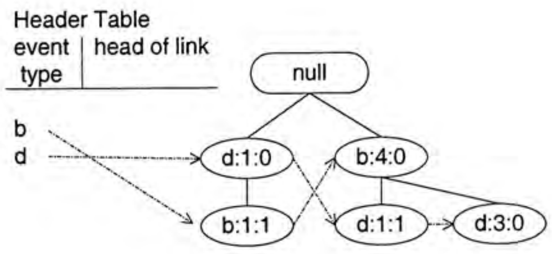


Figure 4.5: x's conditional tree.

Recursive processing: We call the mining procedure recursively, passing along x's conditional DE-tree. The paths containing the event type at the bottom of header table, d, are traversed: $<(b:4),(d:1)>, <(b:4,d:3),(\phi)>, <(d:1),(b:1)>$. The counts in each path are adjusted and counted. The paths become $<(b:1),(\phi)>$ $, <(b:3),(\phi)>, <(b:1),(\phi)>$ and are dx's conditional paths. And the first-part frequency of b in dx's conditional paths $<b:5>$, thus the frequency of episode bdx is $<bdx:5>$, which is a frequent episode. Since dx's conditional paths contain only one event type and cannot form the episode containing more number of combinations of event types, there is no need to call the mining procedure again. The only frequent episode containing event type x, $<bdx>$, is obtained.

Next element in HT: Then we visit the paths of event type d, which is at the second bottom of the header table, from the original complete tree. The paths are $<(d:1),(b:1,c:1,x:1)>, <(b:5,c:1,x:1),(a:1,d:1)>, <(b:5,a:3,c:1),(d:1,x:1)>, <(b:5,a:2,d:1),(x:1)>, <(b:5,d:1,x:1),(a:1,c:1)>$. The event type x in the paths are ignored because all episodes containing event type x have been explored. The event types in the remainingdays part of the paths are then moved if necessary and the frequencies of event types are counted. We obtain the window frequencies $<a:5,b:6,c:4>$ and the first-part frequencies $<a:4,b:6,c:4>$. The frequent episode is deduced as $<bd:6>$. Then d's

Event Set	Paths visited	Conditional paths	{window frequency, first-part frequency, Frequent episodes}
x	$<(d:1),(b:1,c:1,x:1)>$ $<(b:5,c:1,x:1),(a:1,d:1)>$ $<(b:5,a:3,c:1),(d:1,x:1)>$ $<(b:5,a:3,d:1),(x:1)>$ $<(b:5,d:1,x:1),(a:1,c:1)>$ $<(x:1),(\phi)>$	$<(d:1),(b:1,c:1)>$ $<(b:1,d:1,a:1,c:1),(\phi)>$ $<(b:1,a:1,c:1),(d:1)>$ $<(a:1,b:1,d:1),(\phi)>$ $<(b:1,d:1,a:1,c:1),(\phi)>$	$\{<b:5,d:5,a:3,c:4>$, $<b:4,d:4,a:3,c:3>$, $<\phi>\}$
dx	$<(d:1),(b:1)>$ $<(b:4,d:3),(\phi)>$ $<(b:4),(d:1)>$	$<(b:1),(\phi)>$ $<(b:3),(\phi)>$ $<(b:1),(\phi)>$	$\{<b:5>$, $<b:5>$, $<bdx:5>\}$
d	$<(d:1),(b:1,c:1,x:1)>$ $<(b:5,c:1,x:1),(a:1,d:1)>$ $<(b:5,a:3,c:1,d:1),(x:1)>$ $<(b:5,a:3),(d:1)>$ $<(b:5,a:3,d:1),(x:1)>$ $<(b:5,d:1,x:1),(a:1,c:1)>$	$<(b:1,c:1),(\phi)>$ $<(b:1,c:1,a:1),(\phi)>$ $<(b:3,a:3,c:1),(\phi)>$ $<(b:1,a:1,c:1),(\phi)>$	$\{<a:5,b:6,c:4>$, $<a:4,b:6,c:4>$, $<bd:6>\}$
c	$(d:1),(b:1,c:1)>$ $(a:1,c:1),(b:1)>$ $(b:5,c:1,x:1),(a:1,d:1)>$ $(b:5,a:3,c:1),(d:1)>$ $(b:5,d:1,x:1),(a:1,c:1)>$	$<(\phi),(b:1)>$ $<(a:1,b:1),(\phi)>$ $<(b:1,a:1),(\phi)>$ $<(b:1,a:1),(\phi)>$ $<(b:1),(a:1)>$	$\{<a:4,b:4>$, $<a:3,b:4>$, $<\phi>\}$
a	$<(a:1,c:1),(b:1)>$ $<(b:5,c:1,x:1),(a:1)>$ $<(b:5,a:3,c:1),(d:1)>$ $<(b:5,a:3),(d:1)>$ $<(b:5,a:3,d:1),(x:1)>$ $<(b:5,d:1,x:1),(a:1)>$	$<(b:1),(\phi)>$ $<(b:1),(\phi)>$ $<(b:3),(\phi)>$ $<(b:1),(\phi)>$ $<(b:1),(\phi)>$	$\{<b:6>$, $<b:6>$, $<ba:6>\}$

Table 4.1: The summary of the mining process of Example 2.

conditional tree is built with event types a and b and mining procedure is called recursively, which does not result in frequent episodes.

The third element in HT is c. The paths of c are visited. The paths are $<(d:1),(b:1,c:1)>,<(a:1,c:1),(b:1)>,<(b:5,c:1,x:1),(a:1,d:1)>,<(b:5,a:3,c:1),(d:1)>,<(b:5,d:1,x:1),(a:1,c:1)>$. Similarly the event type x and d are ignored from paths. The first-part frequency is $<a:3,b:4>$ and the window frequency is $<a:4,b:4>$. There is no frequent episode generated and no recursive mining procedure call is needed because the window frequency and the first-part frequency of the two event types are below threshold.

Finally, the paths of event type a are visited: $< (a:1, b:1), (d:1) >$, $< (b:5, c:1, x:1), (a:1) >$, $< (b:5, a:3, c:1), (d:1) >$, $< (b:5, a:3, d:1), (x:1) >$ and $< (b:5, d:1, x:1), (a:1) >$. The first-part frequency and the window frequency are both $< b:6 >$. The frequent episode is $< ba:6 >$. Since there is only one event type, b, in a's conditional paths, the mining process ends when we finish processing all the linked lists in HT.

The mining process is illustrated in Table 4.1. **Note**: in this example, we can demonstrate that the subset of a frequent episode may not be frequent: bdx is a frequent episode, but bx and dx are not frequent. □

4.4.5 Completeness of finding frequent episodes

Theorem 1 All the frequent episodes are discovered by the mining process and no infrequent episode is returned.

Rationale Let the event types in the header table be $e_0, e_1,...,e_n$, in the top-down order of the table. First all the paths of e_n are explored and the frequency of episodes formed by the combinations between e_n and $e_0, e_1,...,e_{n-1}$ are examined. Next the whole paths containing e_{n-1} are traversed and only the event e_n will be ignored in the paths. Thus the frequencies of the episodes of the combinations between e_{n-1} with $e_0, e_1,..,e_{n-1}$ are examined. When we examine e_i's paths, the events $e_{i+1}, e_{i+2}...,e_n$, the episode of which we have examined before, are excluded, so as to eliminate redundant considerations while keeping the search of complete set of episodes. □

4.5 Implementation of DE-Tree

The steps of tree construction can be concluded with the pseudocode in Figure 4.6 and 4.7.

In the previous description in the mining process, we seem to indicate that conditional paths are first recorded somewhere in each iteration. This can be a costly

Input: An event database DB, a minimum support θ and window size w
Output: A tree, $Tree$, storing DB
Method: TreeConstruction(DB, θ, w)

```
Procedure TreeConstruction($DB, \theta, w$)
{
    // First scan of $DB$
    count the frequency of each event type in the firstdays part
    count the frequency of each event type in DB
    For each event type in $DB$
        If window frequency $\geq \theta$; Insert the event into frequent event list $l$
    Sort $l$ in descending order according to the frequency of the first day
    Create a null root node
    // Second scan of $DB$
    For each window size of days of events
        For each event in the first day of window
            If the event is in $l$ then insert into $l1$
        Sort the $l1$ in the order of $l$
        For each event in the remainingdays part
            If the event is in $l$ but not in $l1$ then insert into $l2$
        Sort the $l2$ in the order of $l$
        If $l1$ is not null
            Concatenate $l1$ with $l2$
            UpdateTree($Tree, l1, |l1| - |l2|, |l1|, 0, 1$)
}
```

Figure 4.6: DE-tree construction main program.

```
UpdateTree: To insert a window of $n$ events, $l1$, into the tree, $Tree$ with count $c$
Input: The number of events in the firstdays part, $n1$
    The number of events inserted, $i$, which set to 0 initially

Procedure UpdateTree($node, l1, n1, n, i, c$)
{ If ($i == n$) then return // if all events are added, then return
  If $node$ is null // For empty tree
      Create a child node of $l1[0]$, $new\_node$
      $new\_node.count = c$; $new\_node.type =$ firstdays part; $node = new\_node$
      Connect $new\_node$ to the link from header table
      UpdateTree($child, l1, n1, n, i + 1, c$)
  Else
      If the event and the type of one of children of $node$, $child$, are same with that of $l1[i]$
          $child.count$++; UpdateTree($child, l1, n1, n, i + 1, c$)
      Else
          Create a child node of $l1[i]$, $new\_node$; $new\_node.count = c$
          If ($i < n1$) then $new\_node.type =$ firstdays part
          Else $new\_node.type =$ remainingdays part
          Connect $node$ with $new\_node$; Connect $new\_node$ to the link from header table
          UpdateTree($new\_node, l1, n1, n, i + 1, c$)
}
```

Figure 4.7: DE-tree construction.

approach. In our implementation we do not keep the set of conditional paths separately. Instead, when we traverse a linked list from a header table, we do it twice. The first time is to collect the counts for both window frequencies and first-part frequencies. The second pass is to identify the conditional paths, remove the infrequent event types, and insert the resulting paths into a new conditional DE-tree.

Here we list the pseudocode for some of the important mining process. After we have built the FP-tree T for a database, $Mine(T, null)$ will mine all frequent episodes.

In Figure 4.8, the function *CombineAll(Tree, α)* generates all conditional frequent episodes from a single path conditional DE-tree and combines them with the base α.

In Figure 4.6 and 4.10, the function $UpdateTree(t, l, n_0, n, i, c)$ takes a list of n events l and insert it like a window into the conditional tree t. The first n_0 events in l are the firstdays part, and the increment of the counts of all the events in l is equal to c. i is the number of events inserted to t. In the same figure, $ChildPathSearch(x, a, l, t)$

Method: To mine all frequent episodes which include a set of events α
Input: *Tree* (the tree constructed from an event database) and α
Output: All frequent episodes

```
visited list is a global data structure and is set null initially before the mining
Procedure Mine($Tree$, $\alpha$)
{ If $Tree$ contains only single path
        CombineAll($Tree$, $\alpha$)
  else
     For each event $b$ of the header table (start from the bottom)
        For each path containing event $b$
           If there is at least one event in the firstdays part
              For each node $node$ above event $b$
                 If the $node.event$ does not appear in visited-list
                    $node.count+ = b.count$
                 If $node$ is in the firstdays part
                    first-day frequency of $node.event$ $+= b.count$
                 window frequency of $node.event$ $+= b.count$
              If $b$ is in the firstdays part then CountChildFreq($b$)
        For each event type $r$ in the paths visited
           If $r$'s first-part frequency $\geq$ threshold
              Generate frequent episode $r \cup b \cup \alpha$ with support = $r$'s first-part frequency
           If window frequency $\geq$ threshold
              Insert $r$ into the frequent event list $l$
        If $|l| \geq 2$
           // Construct $(b \cup \alpha)$'s conditional tree $Tree$ only with the event types in $l$
           BuiltCondTree($Tree, b \cup \alpha, l, cond\_tree$)
           Mine($cond\_tree$, $b \cup \alpha$)
        Insert $b$ into visited-list
}
```

Figure 4.8: Mining frequent episodes.

```
Procedure CountChildFreq($node$)
{    For each child of $node$, $child\_node$
     While $child\_node$ is in the firstdays part do CountChildFreq($child\_node$)
     If $child\_node$ is not in the visited-list
         First day frequency of $child\_node.event$ $+=$ $child\_node.count$
         Window frequency of $child\_node.event$ $+=$ $child\_node.count$
         CountChildFreq($child\_node$)
}
```

Figure 4.9: Collecting all frequencies of events in the remainingdays part by depth first search.

buildCondTree: To construct a conditional tree, *cond_tree*
Input: A tree, *Tree*, the paths of which containing an event x to be visited
A list of sorted frequent events, l, relative to event x

```
Procedure buildCondTree($Tree, x, l, cond\_tree$)
{    for each path containing $x$
     if $x$ is in the lower part of path
         for each ancestor node of $x$
             if $x$ is in $l$
                 if the ancestor node is in upper part of path then insert $x$ into $p1$
                 else
                     insert $x$ into $p2$
         sort $p1$, $p2$ according to $l$
         concatenate $p1$ with $p2$
         UpdateTree($cond\_tree, p1, |p1| - |p2|, |p1|, 0, x.count$)
     else
         for each ancestor node of $x$
             if $x$ is in $l$ then insert $x$ into $p1$
         $PathBuilt$ = ChildPathSearch($x, l, cond\_tree$)
         If ($PathBuilt \neq x.count$)
             sort $p1$ according to $l$; UpdateTree($cond\_tree, p1, |p1|, |p1|0, x.count$)
}
```

Figure 4.10: Conditional tree construction.

takes a node x in a tree and performs depth first search to find the nodes for frequent event types under x which are in the remainingdays part. Such event types must be in the event type list l. Nodes in such a traversed path are sorted according to the ordering in l and stored in a, which is then inserted into the conditional tree t as firstdays part (binary bit $= 0$).

ChildPathSearch: To search the frequent events in child paths of x and insert the path to a conditional tree
Input: A list of valid frequent nodes below x, a,and a list of frequent event, l

```
Procedure ChildPathSearch($x, a, l, cond\_tree$)
{ If $x$ has no child node then return 0
   For each child node $c$ of $x$
      If $c$ is in the remainingdays part
         If $c$ is in $l$
            Concatenate $c$ with $a$; $path\_built$ += childPathSearch($c, a, l, cond\_tree$)
            If $x.count - path\_built > 0$
               sort $a$ in the order of $l$; UpdateTree($cond\_tree, a, |a|, |a|, 0, c.count$)
            $path\_built$ += $c.count$
         Else $path\_built$ += childPathSearch($c, a, l, cond\_tree$)
      Else $path\_built$ += childPathSearch($c, a, l, cond\_tree$)
   Return $path\_built$
}
```

Figure 4.11: The code for performing depth first search to find the frequent events under the root event and inserting the tree paths into the conditional tree.

4.6 Method 2: Node-List Event Tree

The structure of the DE-tree has two main weaknesses. Since the tree path is divided into two parts to store the firstdays event types and the remainingdays event types separately, the nodes with the same event type in the different parts of tree cannot be shared and it cannot take the full advantage of prefix tree, when the firstdays part is different even if the remainingdays part are the same. The second disadvantage is the cost of depth-first search in the mining process. Thus an alternate method of mining frequent episode is proposed.

The second method also applies a tree to store windows of event types and mine the frequent episodes. It is called a **Node-List Event Tree**(NE-tree). The root of the NE-tree is a null node. Each nodes consists of an event type and a count, the latter of which has different meaning from the count in the DE-tree. The basic structure of the path in the NE-tree is the same as the FP-tree in that the paths are not divided into upper and lower part to store the two parts of window. On the contrary, the event types in a whole window are stored in a path in descending order of their window frequencies regardless of the their positions in window.

To indicate the positions of the event types in a window, a string of binary bits, called **position bits**, is stored in a **window node**, which is attached to the last node of the path representing the corresponding window. Each digit of the position bits indicates a *type* for an event type. The bit is 0 for the event type in the first day of window, whereas it is 1 for the event type in the remaining days of window. The position bits are assigned according to the order of the event types in the path. Thus the most significant bit (MSB) of the position bits indicates the position of the last event type in the path representing the window while the least significant bit (LSB) indicates the location of the top-most event type in the path.

Each window node also contains a **position count**, which stores the number of paths having the same position bits. When there are more than one type of position bits, the window nodes containing the position bits are connected to form a single linked list. The window nodes are arranged in the descending order of the position bits patterns and a pattern with all zero bits will be at the top. The window nodes are sorted because it is more efficient insertion in the tree construction. The reason why zero position bits is at the top is given in Section 4.8.1.

To facilitate the retrieval of the corresponding window nodes of the event nodes so that we need not perform depth-first search to get the window nodes, all window nodes belonging to the same event node are linked to form a doubly linked list. An event node contains a **window pointer**, which directs to its window nodes. For example, a simplified NE-tree is shown in Figure 4.12 in which the rounded rectangle nodes are the window nodes and the capital letter denoted as a group of the position bits and the position count. Take the path $< s, i, d, c >$ in Figure 4.12 as an example, if only event type s in the first days of window and the count of path is 1, then the window node P is 1110:1. For clarity, the figure only shows the links between the last nodes of the paths and the window nodes and omits those links with the upper nodes. The event node, by default, points to the window node directly under the event node itself, such as nodes d, c and x. If the event node has one or more child nodes that are event nodes, it points to the window node of one of its nearest child node. For example, nodes i, s points to G, and b points to F. Each event node stores a count to indicate the number of consecutive window nodes, which counted from the right of the pointers, containing its position in window in the doubly linked list so that all the window node including the event type can be identified. For example, for node $i : 7$, it points to G, in the doubly linked list of window node, G, P, A, F, E, H and D are nodes that contain window frequencies for the node i. Thus the count of i is 7. When a path is a single path, like the rightmost path of the tree in Figure 4.12, all the window pointers of the event nodes points to the window nodes as shown in Figure 4.13(a).

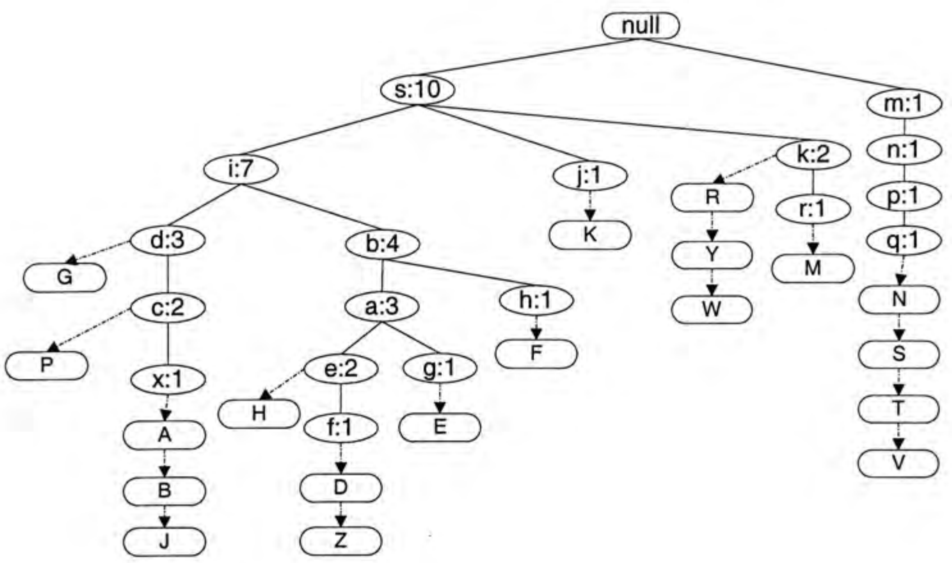
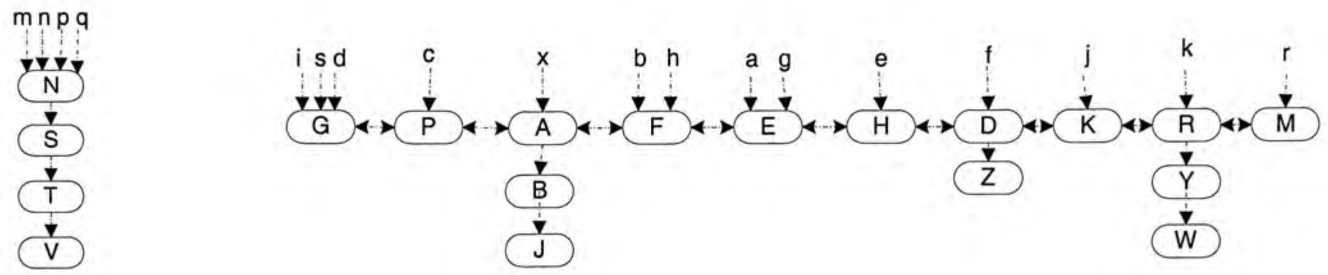


Figure 4.12: A simplified NE-tree.



(a) Rightmost path (b) Child Paths of s

Figure 4.13: Another view of the window nodes in the lists for the tree in Figure 4.12.

Figure 4.13(b) shows a doubly linked list of nodes which are relevant for the s node. The figure also illustrates that the count in the event node is also the number of window nodes under the event nodes, including the window node attached directly to the event node itself. Event node s has the count of 10 because there are 10 columns of window nodes under it. Since only the top node of single linked list, such as A, B and R, appear in the doubly linked list, a column of single linked list is counted as one in calculating the count in the event node.

4.6.1 Tree construction

The construction process of path is the same as that in FP-tree. The steps of obtaining event frequencies from database are similar to that in the DE-tree. Thus we will emphasis explaining the creation of window nodes and doubly linked list.

Example 3 Using the event database in Figure 4.1(a), we construct NE-tree with the same window size and threshold as that in Example 1 to see the difference.

First the event database is scanned to obtain the total window frequency of each event types. The event types of each window are same as that in Example 1 and the window frequency is $< a : 6, b : 7, c : 5, d : 6, m : 4, x : 5, y : 4, z : 3 >$. As before, those event types whose the window frequency below the threshold will be removed. The event types with the threshold are sorted in the descending order of window frequency as $< b, a, d, c, x >$. The order depends on the window frequency instead of the database frequency because a path in NE-tree corresponds to a window in database. The event nodes in path of tree will follow this order when constructing the tree. The sorted frequent event types of each window are shown in Table 4.2 to facilitate the explanation. The second last column of the table contains the sorted event types according to the window frequency, while the last column contains the position bits to indicate the corresponding event type position in the window.

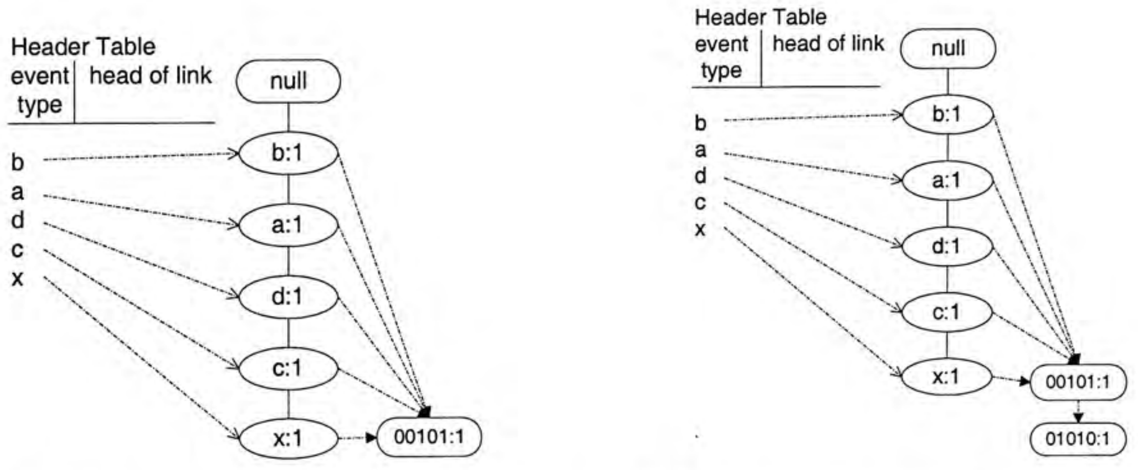
In the eighth window, no event type will be inserted into tree because there is only one event type in the window and a single event type cannot form an episode.

Window No.	Days included included	Partitioned Ordered Frequent Events	Ordered Frequent Events	Position bits
1	1,2	$< (b, a, c), (d, x) >$	$< b, a, d, c, x >$	00101
2	2,3	$< (b, d, x), (a, c) >$	$< b, a, d, c, x >$	01010
3	3,4	$< (a, c), (b) >$	$< b, a, c >$	100
4	4,5	$< (b, a), (d) >$	$< b, a, d >$	001
5	5,6	$< (d), (b, c, x) >$	$< b, d, c, x >$	1011
6	6,7	$< (b, c, x), (a, d) >$	$< b, a, d, c, x >$	01100
7	7,8	$< (b, a, d), (x) >$	$< b, a, d, x >$	0001
8	8	$< (x), () >$	$< \phi >$	-

Table 4.2: The frequent event types included in windows in Example 3.

A null root node is created. The event database is scanned for the second time. The event types in the first two days are read, ie. $< (a, b, c), (y, m, b, d, x) >$. The frequent event types in the two days are identified as $< (a, b, c), (d, x) >$ and sorted as $< b, a, d, c, x >$ while recognizing the position of the event type. Since a, b, c are in the first day of window and d, x are in the other days of window, by mapping 0 to a, b, c and 1 to d, x in the sorted event types $< b, a, d, c, x >$, the position bits of the first window is 10100. For the null root node which has no child node, a new path is of $< (b : 1), (a : 1), (d : 1), (c : 1), (x : 1) >$ is created for the first window. The numbers behind the colon and the event types in the path are the numbers of window nodes representing the event types in the doubly linked list. The last node of path,

x, is also attached a window node of 10100:1 for the event type position information of the whole path. A header table with the sorted event type $< b, a, d, c, x >$ is built. The new nodes are pointed by the entries in the header table with the same event types. The tree after being inserted the first path is as shown in Figure 4.14(a).



(a) After inserting the first window (b) After inserting the first two windows

Figure 4.14: The trees after being inserted the first and the first two windows.

Then the third day of event types are read to create the second window. After selecting and sorting the frequent event types in the window, the ordered frequent event types of the second window is $< b, a, d, c, x >$ with position bits 01010. Since all the event types of the second window are the same as that of the first window, it shares all the nodes of the first window and add a new window node to x node. For efficiency, the new node is inserted in the head of the single linked list. The counts in the event nodes are still 1, because there is one column of node in the doubly linked list. The tree after being added the second window is shown as Figure 4.14(b).

The event types in the fourth day are read. The sorted frequent event types of the third window are $< b, a, c >$, so the position bits are 001. Since there are common nodes b and a in the existing prefix path, the path of the third window can be built on it. Node c is created with count initialized by 1 and is concatenated to the node a. The window node with position bits 001 and position count 1 is created and pointed by the event nodes b, a and c directly because the level of the new window node are at a higher level than the two existing window nodes. The counts of nodes b and a are incremented to 2 for two columns of window nodes in the doubly linked list containing their position bits. The new window node connects to the head of existing window bit nodes list such that all window nodes can be accessed by b and a easily. The tree storing the three windows is shown in Figure 4.15.

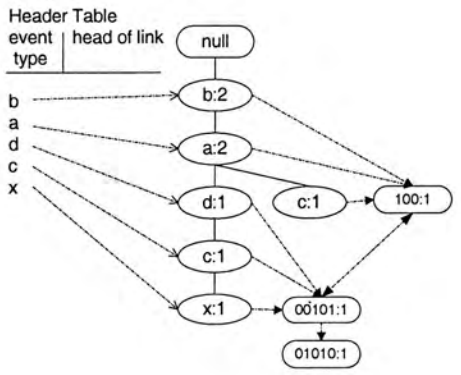


Figure 4.15: The tree after being inserted the three window.

Then the remaining windows are inserted in a similar way. A header table is also created to facilitate tree traversal. The final tree with the header table is shown in Figure 4.16. Figure 4.17 provides a clear view about the relationship of the event nodes and window nodes.

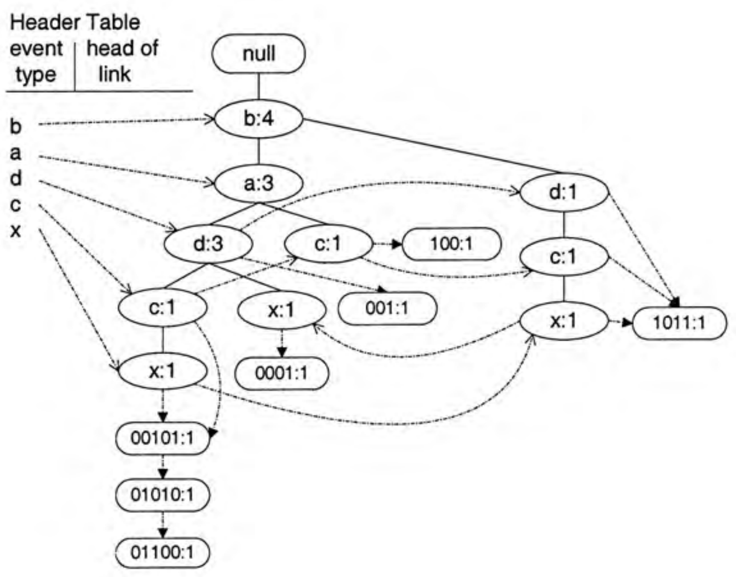


Figure 4.16: The tree after being inserted all windows.

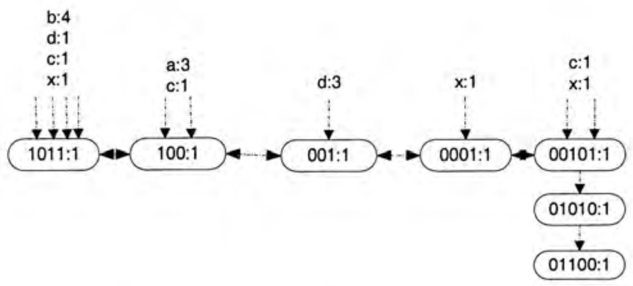


Figure 4.17: Another view of the doubly linked list of window nodes of tree in Figure 4.16.

4.6.2 Order of Position Bits

In our implementation, the order of the position bits is reversed from that we proposed originally. We use a dynamic array of `unsigned integer` to store the position bits. Since one unsigned integer consists of 32 bits, it can only record the position bits for the maximum 32 event types along the path. For a path containing x event types, we allocate an array of $INT((x+32)/32)$ unsigned integers to store the position bits for all event types in the path.

In the mining process, we traverse up the path to visit all the ancestor event types, which leads us to know the levels of the event types in the path. If the bit of the top level event is the leftmost bit of the position bits as proposed in theory, we have to traverse down to collect the total number of events along the path to know the exact digits of the event types. On the contrary, we can directly access the position bits if the rightmost bit is the top level event type.

4.7 Implementation of NE-tree construction

The process of tree construction can be summarized in Figure 4.18, 4.19 and 4.20.

Method: To construct a NE-tree, *Tree*, given a database *DB*
Input: A minimum support threshold θ and a window size w

```
Procedure TreeConstruction($DB, \theta, w$)
{ Count the window frequency of each event type
  Sort the event types in the descending order of the window frequency as $l$
  for each window size of days in $DB$
      Select all frequent event types in window
      Sort the list of event types according to $l$ as $s$
      Compose the position bits, $b$
      UpdateTree($node, s, b, 1, 0, |s|$)
}
```

Figure 4.18: NE-tree construction

In the first scan of database, there is no need to accumulate the event type frequency in database because it is used to sort the event types in descending order for the upper part of tree in the DE-tree. However, in NE-tree, the orders of the event nodes in a path only depend on the frequency of the event nodes appearing in the paths of tree in order to make the tree compact. Thus we only have to calculate the window frequency.

The main idea of *UpdateTree()* is similar to that in DE-tree, except for handling the position bits and the insertion of window nodes into linked list.

When a new event node is created, the path under it should be a single path. Thus

UpdateTree: To insert a window of n event types l with position bits b and position count c into NE-tree
Input: Root of tree, *node*, the path will be built under
The number of event types inserted, i, which is set to 0 initially,
Output: The head window node of the single linked list updated, *win_node*.

```
Procedure UpdateTree($node, l, b, c, i, n$)
{
    If ($i = n$) // if all event types are added, then process the window bits
        CreateWindowNode($b, c, node$)
    If $node$ is null // For empty tree
        Create a new child node of $node$, $new\_node$ with $count = 1$
        Connect $new\_node$ to $node$ and the link from header table
        $new\_node.winptr$ = UpdateTree($new\_node, l, b, c, i+1, n$)
    Else
        If one of children of $node$, $child$, is the same as that of $l[i]$
            $win\_node$ = UpdateTree($child, l, b, c, i+1, n$)
            If a new single list is created in UpdateTree() returned
                If $win\_node$ is not yet inserted to the doubly linked list
                    If $child$ has no its own window node
                        Insert $win\_node$ to the left of $s[i].win$; $child$ points to $win\_node$
                    Else
                        Insert $win\_node$ to the right of $s[i].win$
                Else
                    // re-position the window pointer of node
                    If $child.win$ points to the right of the $win\_node$
                        $child.win = win\_node$
                $child.count{+}{+}$
                Return $win\_node$
            Else // no new single list created
                Return $win\_node$
        Else // no existing child matched
            Create a new child node of $child$, $new\_node$, with $count = 1$
            Connect $node$ with $new\_node$ and the link from header table
            $win\_node$ = UpdateTree($new\_node, l, b, c, i+1, n$)
            $new\_node.winptr = win\_node$
            Return $win\_node$
}
```

Figure 4.19: Tree path insertion for NE-tree.

CreateWindowNode: To create a window node by given a list of position bits b and the corresponding position counts c
Input: The node to be concatenated with the window node, *node*.
Output: A window node, *win_node*, and it is inserted into single linked list if necessary

```
Procedure CreateWindowNode($b, c, node$)
{ If $node$ is newly created and has no window node OR
        $node$ is an old node but has no window node directly under it
        Create and initialize the window node, $win\_node$
        Return $win\_node$
    Else
        // $node$ has its window nodes directly under it
        Insert $b$ into $node.win$ single linked list
        Return $win\_node$
}
```

Figure 4.20: Window nodes creation.

the new nodes along the single path all point to the new window node. However, when the paths share the existing child nodes, the common child nodes at the lowest level moves its position pointer. And the new window node will be inserted in the doubly linked list to the left of the current window node that the shared event node is pointing to. Then all the window pointers of the nodes at a higher level in the same path will move to the new window node if the window node they are pointing to is at a level lower than the new one.

When a column of nodes is inserted to the doubly linked list, the counts of the common event nodes will be increased.

4.7.1 Complexity of NE-Tree Construction

Same as the DE-Tree, the whole tree construction process needs two scan. The first scan is to calculate the window frequency of event types and the second scan is to read and insert the window of event types into the tree. The cost of tree construction is same as DE-Tree and is $O(nm)$, where n is the number of windows and m is the total number of event types in a window. The cost of inserting a path is $O(p)$ where

p is the number of frequent event types in the window. The main cost of inserting a window node into the doubly link list is $O(w)$, where w is the number of window nodes in the single linked list when a window node is inserted for the same path. The height of tree is the maximum number of frequent event types among the windows. The maximum size of tree is the number of total frequent event types in all windows.

4.8 Mining Frequent Episodes with NE-tree

The process of mining frequent episodes from a NE-tree is the same as that of mining from FP-tree, except that we have to follow the linked lists to get the position bits and process the bits when moving the event types from the remaining days of window to the first day.

The position bits will be processed as follows. When mining paths containing an event type x, we follow the x's window pointer and trasverse from left to right to visit the columns of x's window nodes in the doubly linked list according to the x's position count. If x is at the level l, the position bit of x is the l-th bit of the position bits counted from left to right, where $0 < l \leq$ length of x's path. If the position bit of x is 0, it means that x is in the first day of window and all event types in the remainingdays part of window can be moved to the firstday part. In other words, all the event types will be in the first day of window after relocation. Thus the position bits can be reset to zero to indicate all event types are in the first day of window. On the contrary, if the position bit of x is 1, x is in the second part of window and the position bits remains unchanged.

4.8.1 Conditional NE-Tree

Similiar to the DE-tree, for the prefix paths of h, we insert those event types with their window frequencies no less than the support threshold into a conditional database DB', which is then sorted in descending order of window frequencies. A h's conditional tree T' from DB' is constructed. The event types in HT' of T' follows the order of event types in DB'.

The mining process is called recursively with the tree T' and a base event $\alpha = \{h\}$. We repeat the mining phase by following the links from header table and processing the position bits. The frequent episodes formed for each event type h' in the T''s header table are concatenated with the base events $h' \cup \alpha$.

In forming the new conditional tree, we re-compose the new position bits from the conditional path, which may be reset to zero if the base event is in the first day of window.

Remarks: Zero position bits becomes more common in the subsequent conditional trees. Therefore we sort the single linked list in ascending order of position bits and keep the zero position bits at the beginning of the list for facilitating us to update the position count of zero position bits.

4.8.2 Single Path Conditional NE-Tree

When the tree has a single path, we can stop to built conditional tree. Since a single path may contain different position bits and window nodes in different level of tree, we cannot generate all combinations of episodes for all event types in the path directly. On the contrary, we calculate the frequencies of episodes for each event type in the single path.

However, when there is only one pattern of position bits for a prefix path of a base event, we can directly generate all combinations of episodes containing the base event according to the position bits. If the only position bits is zero or all other prefix event types are in the first day of window, all combinations of episodes containing the base event can be formed since all of them satisfy the requirement. When the single prefix path of a base event contains only one position bits in which the base event is in the remaining days of window but there exist at least one event type of the prefix path in the first day of window, the resulting episodes are all the combinations containing the base events and at least one event type in the first day of window.

4.8.3 Complexity of Mining Frequent Episodes with NE-Tree

Same as the DE-tree, the complexity of mining frequent episodes depends on the number of paths and the height of tree. It is a NP-hard problem.

4.8.4 An Example

Example 4: We explain the mining process with the tree constructed in Example 3. Starting from the event type at the bottom of the header table, we traverse all the prefix paths containing the event type x by following the links of x from the header table. After visiting the ancestor event types for each x's prefix path, the corresponding position bits in the single linked list are also obtained. Then three paths of x are get: $< b, a, d, c, x >$(00101:1,01010:1,01100:1), $< b, d, c, x >$(1011:1) and $< b, a, d, x >$(0001:1).

The window frequency and the first-part frequency of each ancestor event type are accumulated from the position bits. The fifth bit of each group of position bits is checked. If the bit is 0, the window frequencies and the first-part frequencies of all ancestor event types will be incremented. When the bit is 1, the first-part frequency of an ancestor event type will be accumulated by 1 only if the position bit of that event type is 0. The window frequencies of all ancestor event types are also increased by 1 provided that all the bits before the position bits of x are not 1. For the first window node 00101 traversed for x, the window frequencies and the first-part frequencies obtained are $< b : 1, a : 1, d : 1, c : 1 >$ and $< b : 1, a : 1, d : 0, c : 1 >$ respectively. The remaining five position bits are checked in the similar way and the final window frequencies and the first-part frequencies are $< b : 5, d : 5, a : 4, c : 4 >$ and $< b : 4, d : 4, a : 4, c : 2 >$. Since the first-part frequencies of all event types are lower than the support threshold, the 2-event episodes of event $x < bx : 4, dx : 4, cx : 2, ax : 4 >$ are not frequent.

Conditional paths and conditional NE-tree: Since only the window frequencies of event types b and d meet the support threshold, we construct a x's conditional tree with two event types b and d. All the prefix paths and the window nodes of event type x are trasversed again to compose the new position bits. The position bits of event types b and d are checked. The new position bits for the first column of the three window nodes are 01, 00 and 01. The x's conditional tree constructed is shown in 4.21.

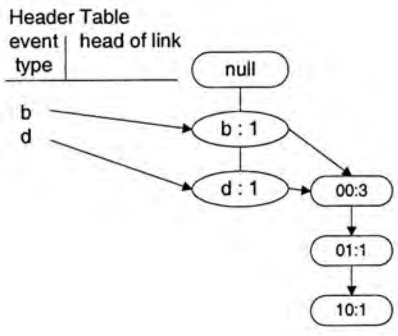


Figure 4.21: The x's conditional tree.

Recursive processing: The mining procedure is called recursively with passing the x's conditional tree. Since the tree contains a single path $< b : 5, d : 5 >$ $(00 : 3, 01, 10)$, we can generate all combinations containing at least two event types, that is $< bdx : 5 >$. The search for episodes consisting of event type x ends here.

Next element in header table: The prefix paths and the window nodes of the second bottom event type c are traversed: $< b, a, d, c >$(00110:1,01010:1,01100:1), $< b, a, c >$(100:1) and $< b, d, c >$(1011:1). The fourth bit of position bits are checked for accumulating the window frequencies and the first-part frequencies as $< b : 5, a : 4, d : 4 >$ and $< b : 4, a : 3, d : 4 >$. From the first-part frequencies, we can know that there is no frequent episode of event type c obtained. Since there is only one frequent event type b whose window frequency is no less than the support threshold, there is no need to construct c's conditional tree because we have already obtained the frequency of episode $< bc >$. Thus the generation of all episode containing event type c is terminated.

The mining process for the other event types in header table is in the similar way.

The process is finished when there is only one event type b remaining unvisited in the header table.

The mining process can be concluded in Table 4.3.

Event set	Paths visited	{Window frequency, First-day frequency}	Frequent Episode
x	$< b:4, a:3, d:3, c:1 >$ (00101:1,01010:1,01100:1) $< b:4, d:1, c:1 >$(1011:1) $< b:4, a:3, d:3 >$(0001:1)	$\{< b:5, d:5, a:4, c:4 >, < b:4, d:4, a:4, c:2 >\}$	ϕ
dx	$< b:1 >$(00:3,01:1,10:1)	$\{< b:5 >, < b:5 >\}$	$< bdx:5 >$
c	$< b:4, a:3, d:3, c:1 >$ (00101:1,01010:1,01100:1) $< b:4, a:3 >$(100:1) $< b:4, d:1 >$(1011:1)	$\{< b:5, a:4, d:4 >, < b:4, a:3, d:4 >\}$	ϕ
d	$< b:4, a:3 >$ (001:1,0001:1,00101:1,01010:1,01100:1) $< b:4 >$(1011:1)	$\{< b:6, a:5 >, < b:6, a:4 >\}$	$< bd:6 >$
a	$< b:4 >$ (1011:1), (100:1),(001:1),(0001:1) (00101:1,01010:1,01100:1)	$\{< b:6 >, < b:6 >\}$	$< ba:6 >$

Table 4.3: The summary of mining process in Example 4.

4.9 Performance evaluation

To evaluate the performance of the two proposed methods, we conducted experiment 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.

4.9.1 Synthetic data

The synthetic data sets are generated from a modified version of the synthetic data generator in [28], which produces customer transactions for mining sequential patterns. In [28], a non-empty set of items is an itemset, whereas a set of ordered itemsets is a sequence and a set of transactions for a customer is a customer-sequence. The original data generator gives a set of customer-sequences according to the realistic retailing environment that customer buys a set of items in at least one transaction.

The original generator takes the parameters listed in Table 4.4. The number of transactions for a customer and the number of items in a transaction are in bell-shaped normal distributions with means as $|C|$ and $|T|$ respectively and the small standard deviations.

Parameter	Description
$\|D\|$	Number of Customers
$\|C\|$	Average number of transactions per Customers
$\|T\|$	Average number of items per Transaction
$\|S\|$	Average length of maximal potentially large Sequences
$\|I\|$	Average size of Itemsets in maximal potentially large Itemset
N_S	Number of maximal potentially large Sequences
N_I	Number of maximal potentially large Itemsets
N	Number of items
c	Correlation level

Table 4.4: Parameter settings of the orignial synthetic data generator.

At the begining of the customer-sequences generation, it prepares two tables, (1) N_I potentially large itemsets table and then (2) N_S potentially large sequences table. Two tables are produced in a similar way. In generating potentially large sequences, it first determines the number of itemsets to be contained in the sequence using Poisson distribution with mean set to $|S|$. For the first sequence, all itemsets are selected randomly from the large itemsets table. But in the following sequences, part of the itemsets, the number of which is decided by an exponentially distributed random variable with mean as the correction level, are selected from the previous sequences to take the common items between sequences. Each large sequence in the table is also assigned a weight obtained from an exponential distribution with unit mean, to give a biased sequence selection. The weights are normalized so that the sum of the weights for all the sequences in the table is 1.

Next it is ready to generate $|D|$ customer-sequences. For each customer-sequence, it first determines the number of transactions and the average size of each transaction with the Poisson distributions given means as $|C|$ and $|T|$ respectively, and then chooses a series of potentially large sequences from the table. The large sequences in a customer-sequence are obtained randomly from the table of N_S large sequences according to the probability assigned in the weights.

In the actual usual case, a sequence is not complete that some items may be missed. The generator imitates the situation with a corruption level c, which is fixed and calculated from a normal distribution. When a uniformly distributed random number between 0 and 1 is less than c, an item will be removed from the sequence when inserting a transaction into a sequence.

Our data generator is developed on the original one with the consideration of overlapping windows.

In building our data generator, the large itemset and the large sequence in the original data generator become a set of events in the same day and the frequent episode respectively. Projecting the customer-sequence into the event dataset in our case, we are to produce a dataset consisting a single customer with a number of transactions. The transactions for the only customer become days in the event dataset whereas the items become events. Similarly, a x days of window is a group of x consecutive transactions. Thus a frequent episode corresponds to a large sequence, whereas a frequent episode within a single day corresponds to a large itemset. And we will call the latter frequent eventset. Our data generator takes the nine main parameters listed in Table 4.5.

We first produce the tables of frequent eventset and frequent episodes with the same method as the original one.

Next we assign events to the days. For the first day, we select the events randomly. In the subsequent $(W-1)$ days of the first window, the events are selected according to the events in the previous days by following the method of generating items into the transactions for a customer. The correlation between events and the corruption of episodes are also considered and applied. Then the events in the last $(W-1)$ days

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
$\|E\|$	Average size of frequent eventset	1, 1.75
L	Number of frequent episodes	1000
F	Number of frequent eventset	5000
M	Number of event types	100 - 1000
W	Window size	2 - 10
c	Correlation level	0.25

Table 4.5: Parameter settings of the synthetic data generator.

of the previous window is copied to a window buffer as the first $(W-1)$ days of the second window. The events in the last day of the second window are generated by referring the events in its first $(W-1)$ days, where the second window can be viewed as the second customer-sequence with W transactions. The events in the first $(W-1)$ days of the second window are discarded and only the events in the last days of the window are inserted into the dataset as the $(W+1)$-th day's events. The remaining days of events are generated in the similar ways until a dataset of $|D|$ days of events are produced.

Relative Performance

Four datasets with different parameter settings are shown in Table 4.6 are produced. We perform the experiments under different thresholds and window sizes with datasets D1 and D2 while the datasets D2 to D4 are used to perform the experiment under different number of days and event types.

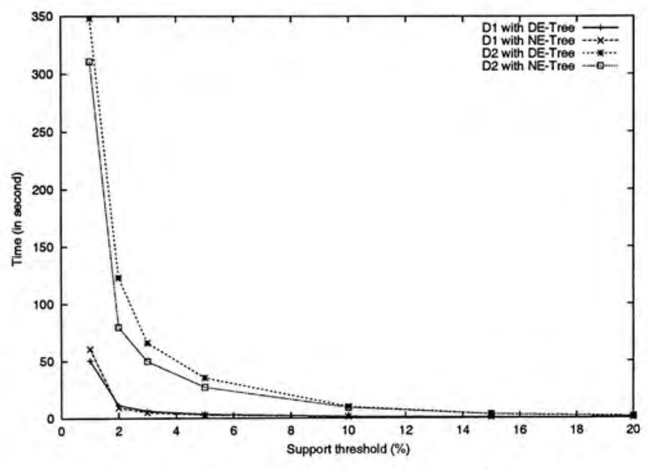
In our implementation, we used linked lists to keep the frequent episodes, one list for each episode size. Each of these lists is kept in an order of decreasing frequencies for a ranked display to the user at the end of the mining. We measure the run time as the total execution time of both CPU time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

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

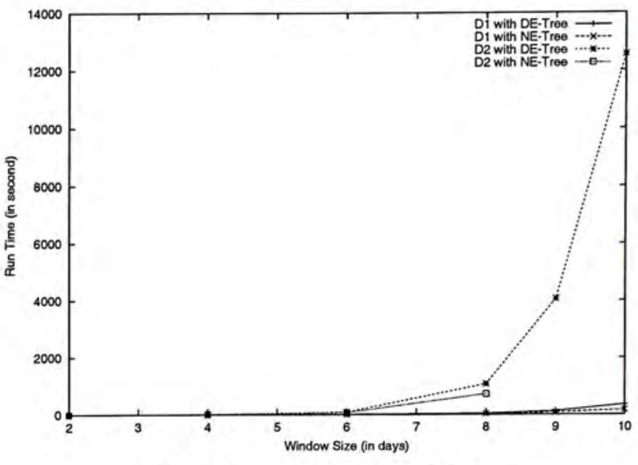
Table 4.6: Four datasets with different parameter settings used in the experiment.

is approximately equal to the mining time because the building time is not significant comparing to the mining time. Among the all experiments conducted, the maximum construction time is less than 2 minutes. Each data points in graphs are the mean time of the several runs of the experiment.

The run time decreases with the support threshold as shown in Figure 4.22 (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 items in the smaller conditional trees. NE-tree performs better than DE-tree especially when the dataset is large. It is because the paths of tree are longer when the threshold is smaller and much time are consumed in depth search to determine the actual counts of items in prefix path. Although DE-tree needs considerable time in performing depth search, the difference of execution times of two trees are not large because NE-tree needs to check each set of position bits one by one to identify the positions of items in windows which is also time consuming.



(a) window size = 3

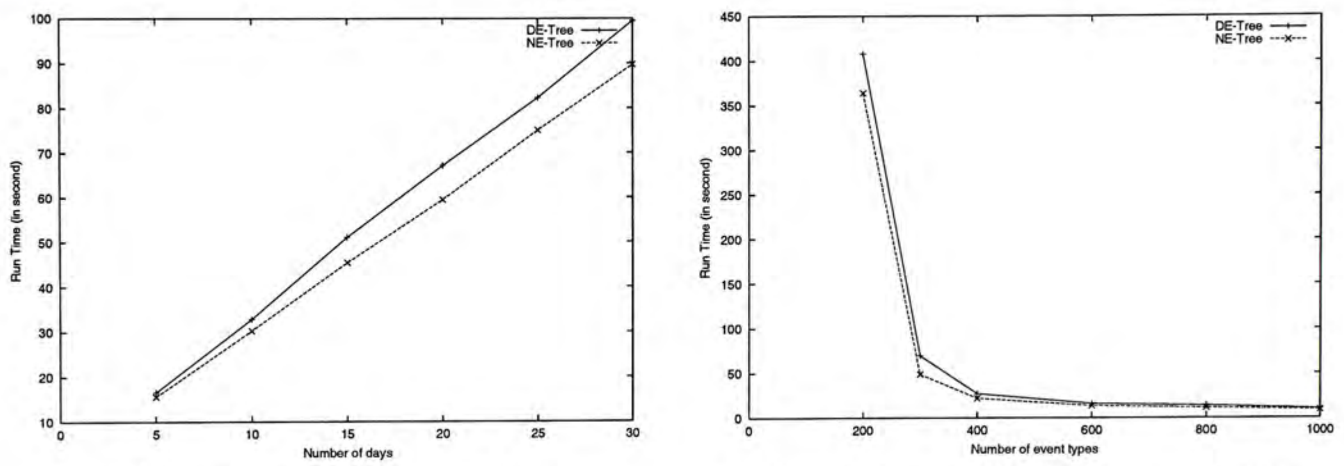


(b) threshold = 20%

Figure 4.22: Performance of synthetic datasets D1 and D2.

Figure 4.22(b) shows the effect of different window sizes on the run time. The dataset 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 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 4.23(a) shows that the execution time increases linearly with the number of days.



(a) varying number of days

(b) varying number of event types

Figure 4.23: Synthetic dataset D2 with window size = 3 and threshold = 10%.

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 a window size of 3 days. The result is shown in Figure 4.23(b).

The curve does not rise as expected, but falls exponentially as the number of event types increases. Since the number of days and the number of event types per day are fixed, 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.

The effect of different numbers of event types with various thresholds on the run time is also evaluated and represented in Figure 4.24. Two datasets, D3 and D4, which have different numbers of event types as 200 and 500 respectively, are used. The run times required for D3 are much more than D4 due to the higher frequencies of event types in D3. When the support threshold is less than 8, the run time for D3 is too large (> 60000 seconds) and cannot be measured.

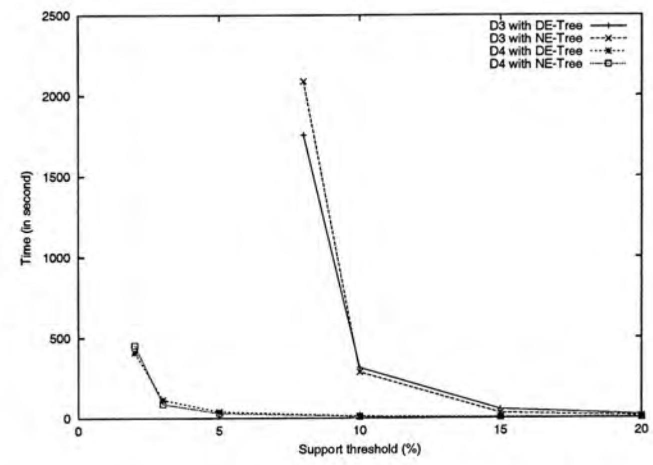


Figure 4.24: Synthetic datasets D3 and D4 with window size = 3.

In usual case, the main memory required for the tree is approximately 40MB. For the extreme case that the program requires long time (> 10000 seconds) to run, the size of the process in the memory is about 70MB.

We also compare the difference between the size of FP-Tree [23] and that of the two event trees we proposed. Two datasets, D2 and D4, are used and the threshold is fixed to 1% and 2% respectively. For the FP-tree, we treat each event type as an item and a database record as a transaction. The frequency of an itemset is the number of records that contains the itemset. The sizes of trees are shown in Table 4.7.

Tree Type	Window Size (in days)	Maximum Size	
		in D2	in D4
FP-Tree	-	3200K	2300K
DE-Tree	2	6300K	4400K
DE-Tree	3	16M	12M
NE-Tree	2	6500K	4500K
NE-Tree	3	15M	12M

Table 4.7: The sizes of trees for two datasets, D2 and D4, with thresholds fixed to 1% and 2% respectively.

The sizes of event trees are larger than the size of FP-Tree. It is because a path in the event tree contain the events from more than one days, while in FP-Tree the path contains the items from only one transaction. The paths in FP-Tree are shorter. So the subsequent conditional trees are smaller and less conditional trees are generated in FP-Tree. However the size difference is still reasonable and acceptable.

The maximum height of tree generated by the event tree are recorded and shown in Figure 4.8. The datasets D1 and D2, which have 10 and 20 average events per days respectively, are used.

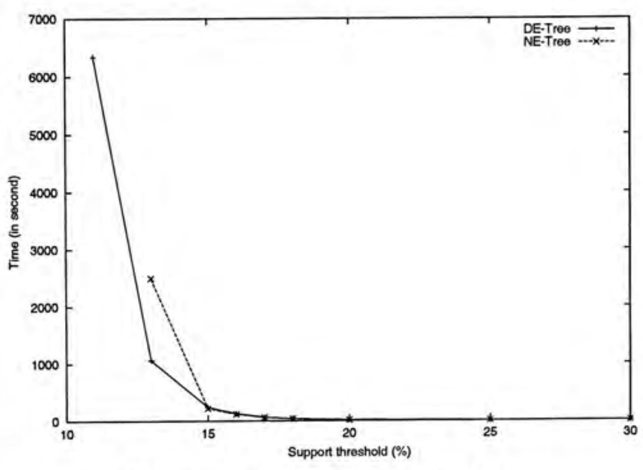
	Window Size (in days)					
	2	4	6	8	9	10
Max. Height in D1	52	44	96	127	152	156
Max. Height in D2	3	62	119	211	223	252

Table 4.8: The maximum height of event tree for different window sizes on dataset D2.

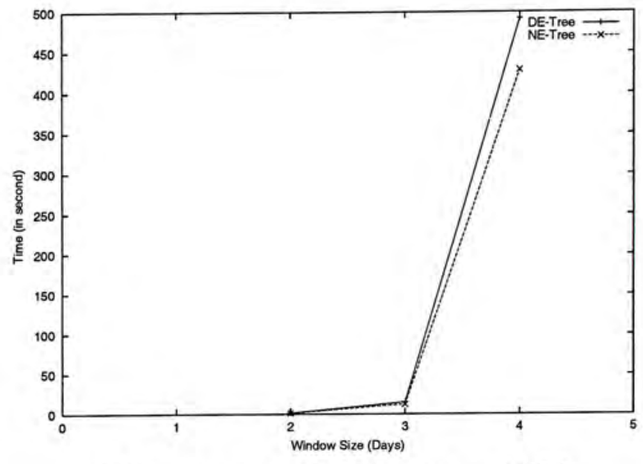
4.9.2 Real data

The real data set is the news event extraction result from a internet repository of a number of local newspapers in Chapter 3. It contains 121 event types and 757 days.

The performance of two trees using the real dataset with different thresholds and window sizes are shown in Figure 4.25(a) and Figure 4.25(b). In Figure 4.25(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 the 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) window size = 3 days



(b) support threshold = 20%

Figure 4.25: Performance of real dataset.

The performance on varying the window sizes are shown in Figure 4.25(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 not be reduced. Thus the mining time increases with the window size.

Real Dataset Results Interpretation

Since the frequency of the events obtained from newspapers are much less than the events of stock price movement, we have set the threshold to 10% to allow more episodes including the newspaper events to be mined. The mining results has 1618500 episodes in total. Most of the frequent event types in the real data set are the events of stock price flat, so the frequent episodes mined are always contained the stock flat events. But it is meaningless to have the episodes the events in which are all 'stock flat', e.g. "the stock of Hong Kong Electric and Dow Jones Index flat within 3 days", although which is the most frequent episode in the result mined. We have selected some interesting episodes mined with threshold set to 10% and window size set to 3 days as shown in Table 4.9. Since we record the events as long as they are mentioned in the newspapers, even though the episodes happen once only, the supports of the events and their episodes may be more than once. So the support of the second last episode in Table 4.9 is 77, although PCCW buys HK Telecom once. 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). Finally, the event of PCCW buys HK Telecom can be seen as a good news for Cheung Kong.

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
PCCW buys HK Telecom, Cheung Kong ups	77
Hutchison buys company, Cheung Kong downs	75

Table 4.9: The frequent episodes selected from the results mined with threshold = 15% and window size = 3 days.

In the whole set of episodes mined, 56 episodes contain the newspaper events only and does not contain any stock events. However, in the episodes containing more than 2 events, they always contain the frequent 'stock flat' events appearing in the 2-event episodes. Because there is no constraint set in the mining process and the method proposed is for general use, the events in some episodes mined are seemed to be unrelated. For example, the episode of 'SKH properties ups and Nasdaq downs within three days" seems does not make sense, because the SKH properties and Nasdaq are in different field and SKH is less likely to be affected by Nasdaq.

As there are rather a lot episodes containing the stock flat events only, to illustrate the actual number of potential useful episodes which contains at least one non stock-flat event, we calculate the number of *interesting episodes* which contains at least one non stock-flat events. Figure 4.26 illustrates the distributions of each size of the total episodes and the interesting episodes mined from the real dataset. It shows that the proportion of the interesting episodes is high in the total episodes mined.

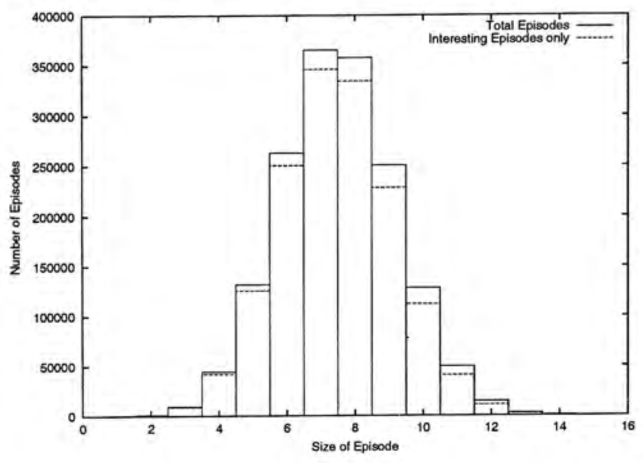


Figure 4.26: Distribution of total episodes and interesting episodes mined from the real dataset.

The maximum heights of the trees generated for the real dataset are also recorded and shown in Table 4.10. The experiment is conducted under various support thresholds and window sizes. In the former case, the window size is set to 3 days, while in the later case the support threshold is set to 10%.

Support Threshold (%)	5	10	15	20	25	30
Max. Height	47	46	43	43	41	38
Window Size (in days)	2	3	4	5	6	7
Max. Height	36	46	52	55	57	58

Table 4.10: The maximum height of event tree for different window sizes on dataset D2.

We have also recorded the maximum size of trees resided in memory when the process generates the 1618500 episodes, that is when the support threshold and the window size are set to 10% and 3 days respectively. The maximum size of memory used is 100MB.

4.10 Conclusion

We consider the problem of mining frequent episodes for financial events. We propose a new definition of frequent episodes which differ from previous work and which amends a deficiency in earlier definition. The new definition does not render the nice property that all subsets of a frequent episode are also frequent. This means that apriori-like methods cannot be used. We have derived and described a new mining method based on a prefix tree structures which bears some similarity to the FP-tree but with distinctive features to make it work for our problem, and proposed a corresponding mining procedure. We have conducted experiments to evaluated the performance of the method. We have found that the performance of the proposed method is efficient and effective.

On the other hand, although the performance of NE-tree on run time is slightly better than DE-tree, the single linked list in NE-tree storing the position bits is not systematic. In the worse case when all position bits exist, there will be 2^n number nodes in the list, where n is the number of events in path. Therefore the single linked list used to store the position bits may be changed to other kind of data structure, such as tree, that can reduce the storage and avoid the considerable memory residence in the worst case.

Chapter 5

Mining N-most Interesting Episodes

In the method introduced in Chapter 4, a threshold is given by users at the beginning to specify the minimum support of the frequent episodes to be mined. However, in many cases, we do not know the frequency distribution of event types in database and the optimal threshold for obtaining the most suitable number of episodes. If the threshold is set too low, the run time is long and a large number of resulting episodes is output that we have to waste time to select the desired frequent episodes. But when the threshold is set too high, there may be too few episodes mined. We usually have to tune the threshold to obtain the appropriate number of episodes. A better alternative to obtain the desired number of episodes is to specify *the number of the most frequent episodes* for all sizes of episodes to be mined, that is the "N" in the problem of finding the N-most interesting k-episodes. It is easier to decide N than the threshold to control the final number of resulting episodes. In this chapter, we propose a method applying the two tree structures discussed in Chapter 4 to find the N-most interesting k-episodes.

5.1 Introduction

In Chapter 4, when we mined episodes from the synthetic data, we have to choose a threshold first. But it is usually difficult to decide a suitable threshold, because we do not know about the frequencies of event types in the database. In general, we try a large threshold in the beginning, which, however, results in no episodes or too few episodes mined. Then the smaller threshold is set but the execution time is long and too much episodes mined. In this case, we also have to search manually for our actual desired episodes from the results mined. To facilitate getting the desired number of results, the number of the top-most frequent episodes, rather than the support threshold, is specified.

[29] and [30] have proposed the methods for mining N-most interesting itemsets. While the former, *Itemset-Loop* [29] , bases on the classic sequential pattern mining algorithm, Apriori generation [19] , the later, BOMO, bases on the FP-tree algorithm in [23].

Starting from the potential 1-itemset, Itemset-Loop [29] uses the k-itemset mined to generate the candidate $(k+1)$-itemset which in turn generates the potential $(k+1)$-itemset. Itemset-Loop also adjusts the support threshold in each iteration of candidate k-itemset. To avoid missing interesting itemset, the support of the N-th potential $(k+1)$-itemset is checked. When the N-th support of the sorted candidate $(k+1)$-itemset is less than that of the potential k-itemset, it implies that some $(k+1)$-itemset which containing the itemset are missed in the potential $(k+1)$-itemset. It loops back to find the new potential k-itemset whose support meets the support of the N-th of candidate k-itemset. The main idea of Itemset-Loop is good. But since the core of the method is the Apriori-gen, its performance is limited due to the repetitive scan of database.

BOMO [30] applies the FP-tree algorithm to mine the N-most interesting itemsets. It assigns a threshold for each size of itemset. All threshold is set to zero initially. Once the N-th frequent itemset is found and its support is greater than the current threshold of the size of itemset, the corresponding threshold is updated. The threshold for constructing subsequent conditional trees is the minimum value among the thresholds of all sizes of itemsets.

To increase the efficiency, it evaluates the initial thresholds after building the initial complete tree. It transverses down the paths of the complete tree to the level which is equal to the maximum size of itemset the user target and records the supports of the item nodes in each level. The initial threshold for the k-itemset is the N-th support of the support list at the k-level sorted in descending order.

BOMO has shown a satisfactory performance. Thus we have adopted BOMO to the two trees in the previous chapter to mine the N-most interesting episodes.

5.2 Method

The method of mining N-most interesting k-episodes we proposed is a variation of the mining frequent episodes approaches with two trees in Chapter 4. The variation is based on [30] with some significant improvement to speed up the mining process.

Here is the formal definition of the N-most interesting k-episodes:

Definition The **N-most interesting k-episodes** is the top N episodes in the list of k-episode sorted in descending order of support, where $2 \leq k \leq k_{max}$ and k_{max} is the maximum limit of episode size assigned by user to be mined.

In the mining method proposed in Chapter 4, a support threshold is assigned to state the lower bound of support of the frequent episodes to be mined in each running time. The other purpose of the threshold is to recognize the frequent event types and their subsequent frequent episodes, removing the less frequent event types during the mining process. Thus the threshold can also reduce the size of conditional trees and shorten the mining time.

On the contrary, in mining N-most interesting k-episodes, only the number of top episodes, N, is given, instead of the support threshold, to describe the target frequent episodes. Taking the advantages of support threshold for excluding those less frequent event types and reducing the mining time, we adopt the method proposed in Chapter 4 by setting and varying the thresholds which are implied by the corresponding supports of the N-th frequent k-episode. The detail of the methods is described as follows:

Initial Threshold

Because there is no threshold assigned initially, we set all the thresholds to 1. The threshold is set to 1 instead of 0 because, in our definition, the frequent episodes are those episodes the supports of which are no less than the threshold. And if the threshold is set to 0, it may imply that even if the event type does not exist in the database, the event type can be included in the frequent episodes and the conditional trees. So it would be more reasonable to set the initial threshold to 1.

Individual and Dynamic Support Threshold

Our task is to find the N-most interesting episodes for each target episode size. Since the minimum support among the N-most interesting episodes for each size of episodes may be different, each size of episode is assigned its own *individual and dynamic support threshold*, ξ_i, where $2 \leq i \leq k_{max}$. The threshold has the same meaning with that in the approaches in Chapter 4, but the thresholds in this problem are continuously updated whereas the threshold is fixed in the previous problem. Each individual threshold is set to one initially. During the mining process when the support of a k-episode is no less than ξ_k, the k-episode is inserted into the final k-episode result list in descending order of support. The individual thresholds ξ_k is upgraded to be the support of the N-th most highest support among all the k-episodes discovered so far if the support of the N-th most frequent k-episodes is greater than the current threshold.

Minimum Threshold

The individual threshold ξ_k is only to determine the frequent k-episodes. When deciding frequent event types to generate conditional trees, another threshold is used.

When selecting frequent event types for constructing conditional tree, the supports of the event types along all the prefix paths of the base event types are accumulated. The frequent event types are those event types whose supports meet the threshold ξ_{min}, where ξ_{min} is the *minimum threshold* among the set of thresholds for the different sizes of episodes to be included in the next conditional tree. That is, for a α's conditional tree, when determining ξ_{min}, for generating next $(\alpha \cup \beta)$'s conditional tree,

$$\xi_{min} = min(\xi_{j_{min}}, \xi_{j_{min}+1}, ..., \xi_{j_{max}}) \tag{5.1}$$

where

$$\begin{aligned} j_{min} &= |\alpha| + 3 \\ j_{max} &= |\alpha| + |\text{longest prefix path of } \beta| + 1 \end{aligned}$$

and where ξ_i, $j_{min} \leq i \leq j_{max}$, is the threshold for the i-episodes and ξ_{min} is the threshold for selecting the event types in generating the conditional trees. When generating $(\alpha \cup \beta)$'s conditional tree from the α's conditional tree, the possible sizes of episodes that the new conditional tree can produce are between $|\alpha| + 3$ and $|\alpha| + |\text{the longest prefix path of } \beta| + 1$. And the event types contained in the α's conditional tree can appear in any size of episodes. The frequencies of the event types should meet the thresholds of all the possible sizes of episodes that the tree can generated. Therefore the support threshold for selecting the event types to construct the conditional tree is the minimum threshold among the thresholds of all size for the possible episode can be generated as shown in equation 5.1. The values of j_{min} and j_{max} vary with the conditional trees to be generated.

5.2.1 Threshold Improvement

Basically, we may set all thresholds ξ_k to 1 initially and increase the threshold ξ_k once the support of the N-th k-episode mined is greater than the current threshold ξ_k. But the mining time needed is long and it is not efficient especially when the size of conditional tree is large. Thus we propose an improvement to push the initial threshold, reducing the size of the initial conditional tree. The main idea of the improvement is to increase the threshold as high as possible before the start of the mining process.

Top-down Scan of Header Table

Since the more frequent an event type is, the higher the supports of its episodes are, we start from the second top event type in the header table to mine the more frequent episodes first. The episodes with high supports can push the thresholds up in the beginning of the mining process, less frequent event types will be discarded and this will reduce the subsequent conditional tree sizes.

Evaluation of Initial Individual Threshold

When we have finished mining episodes for a base event type e_i, where $0 \leq i <$ total number of event types in the initial complete tree and i is the position of the event type in the header table of the initial tree, the possible individual thresholds can be updated are $\xi_2, \xi_3, ..., \xi_{i+1}$. Next we find the episodes associated with the event types e_{i+1}, the range of size of which is $\xi_2, \xi_3, ..., \xi_{i+1}, \xi_{i+2}$. But since there is no $(i+2)$-episode mined, the individual threshold for $(i+2)$-episode, ξ_{i+2}, is 1. Although we have tried to raise the thresholds, the minimum threshold ξ_{min} is still 1 because ξ_{i+2} is minimum and is 1. Hence in addition to improving the individual thresholds with top-down scan approach, we need to evaluate the initial thresholds.

Since the support of a node x at level k implies the support of the k-episode formed by the event type x and all its ancestor event types in the prefix path. Once we have built the initial complete tree, we perform depth-first search to find and record the support of each node at the level within the maximum size of episode user specified. Then the supports of the nodes at each level are sorted in descending order and the initial individual thresholds are the N-th support of the event nodes at the corresponding level.

The evaluation method can be applied to both tree structures directly. For tree 1, the nodes at the top levels of the tree are those event types in the first day of window. When we retrieve the first k events nodes as a k-episodes, the episodes must be valid since the paths are inserted to the tree only if at least one event type is in the first day of window. Therefore we can simply retrieve the support of k-level node to obtain the N-th k-episodes.

On the contrary, we cannot apply the method on NE-tree directly. In NE-tree, the level the event type situated at is according to the window frequency regardless of the event type position in window. Using the method applied in DE-tree to evaluate the initial individual thresholds, we may also include those invalid episodes that no event type is in the first day of window and inaccurate results are produced.

In calculating the episode frequency, we have to check if the current base event type or at least one of all its ancestor event types is in the first day of window. To facilitate the retrieval of the count of valid episode, each event node contains a '*valid episode count*' similar to the count in node of DE-tree which stores the number of valid episodes containing the current event type and all its ancestor event types. The valid episode counts are inserted to the nodes during tree construction. For example, given three position bits for a single path, 1011, 00111 and 000, the valid episode counts of the nodes from top to down in order are 1, 1, 2, 2 and 1.

Evaluation of the complete path

The minimum threshold ξ_{min} is the smallest value among the individual thresholds ξ_i for different sizes of itemsets. However the smallest value usually is the support of the largest size of episode. It is because the smaller episodes are usually the subsets of the larger episodes. Therefore, the minimum threshold is usually the support of the largest episode.

For an event type e_i in b's conditional tree, where i is the position of e_i in the header table of b's conditional tree, $0 \leq i <$ number of event types in b's conditional tree and b is a set of base event types, a *complete path* of e_i is a prefix path which contains e_i and all e_i's i ancestor event nodes. When accumulating the frequencies of all event types in the prefix paths of an event type e_i for constructing $(e_i \cup b)$'s conditional tree, if e_i's longest prefix path is a complete path, the maximum size of episode, j_{max}, that the conditional tree can generate is $i + 1 + |b|$. It also implies that all subsequent minimum thresholds ξ_{min} for constructing its subsequent conditional trees of $e_i \cup b$ are the thresholds for the complete path $\xi_{i+1+|b|}$ if all ancestor event types are reserved. Thus the minimum thresholds ξ_{min} keeps the smallest support values $\xi_{i+1+|b|}$ in the subsequent mining of episodes containing the event types e_i and b . However the minimum threshold for the maximum size of episodes is used to reserve the event types for the largest episode with size $i + 1 + |b|$ mined at the end of the mining process of the conditional tree. Therefore the small minimum threshold for the longest path is a bottleneck of increasing the minimum threshold in the mining process.

Pre-insertion

To improve the minimum threshold, we pre-insert the j_{max}-episode of the complete path, $e_0 \cup e_1 \cup ... \cup e_i \cup b$, into the current N-most j_{max}-episode result list if its support is no less than the threshold $\xi_{j_{max}}$, which is the N-th support of the k-episode mined so far. The maximum size of the episodes the subsequent $(e_i \cup b)$'s conditional tree generates will be smaller than $i + 1 + |b|$ and the j_{max}-episode will not be inserted to the result list again. In determining the minimum thresholds in the subsequent conditional trees, there is no need to consider the threshold of the episodes with the size of complete path, $\xi_{j_{max}}$. Thus the maximum size of episodes, j_{max} the conditional path can produce is

$$|b| + 3 \leq j_{max} < |b| + |complete\_path|$$

However in counting the total window frequencies of event types in tree, although the episode containing the complete path is inserted into the final result set, the frequencies of all the event types in the complete path are still counted and considered in constructing the next conditional tree because the path of event types consists of the sets of smaller sizes of episodes.

We only pre-insert episodes of the complete path only. For the other smaller sizes of episodes, since there are more than one paths in tree containing the same episodes, it is time consuming and is not cost effective to count the supports of the episodes.

In identifying the complete path of the current event type, the number of event nodes in the prefix path should be equal to the position of the event type in the header table. When applying the improvement method to NE-tree, the frequency of the event type e_i's complete path is the valid episode count of the last node of path, that is the count of the event type e_i, because the complete path is a unique path in tree.

However in DE-tree, the prefix path for the complete path may not be unique. Since there may be some event types, e_k, where $k > i$, under the current event node of e_i in the first part of path separating the ancestor event types in the second part of paths, the frequency of the complete path is the total count of the all complete paths in the tree.

5.2.2 Pseudocode

The overall steps of mining the N-most interesting k-episodes is similar to that in the Chapter 4. The pseudocode of mining the N-most interesting k-episodes is shown in Figure 5.1 and 5.2. The codes are generalized to suit the two tree structures.

5.3 Experimental Results

We compare the performance of two trees mining N-most frequent episodes with the two original trees proposed in Chapter 4 mining episodes with a given *optimal threshold*. The optimal threshold assigned for comparing the performance of the corresponding N-most trees is the minimum support among the supports of all sizes of N-most frequent episodes mined by the N-most trees. Since the average support

N-most_Freq_Episode: To mine the N-most frequent episodes.
Input: The maximum size of episodes mined, max,
Output: N-most frequent k-episodes, where $2 \leq k \leq max$.

```
Procedure N-most_Freq_Episode()
{ Let $c\_max$ be the max size of episodes in the current conditional tree
  $c\_max = 0$; $\xi_{min} = \xi_0 = \xi_1 = ... = \xi_{max} = 1$
  Construct initial complete tree, $tree$, with $\xi_{min}$
  // pre-evaluate the initial individual threshold, $\xi_0, \xi_1, ..., \xi_{max}$
  for each level $l \leq max$ of $tree$
    for each child node $e$
      if $e.count \geq \xi_l$
        insert $e.count$ into $support\_list[l]$
        update $\xi_l$ as $N$-th support in $support\_list[l]$
  Mine($tree, null, \xi_0, \xi_1, ..., \xi_{max}$)
}
```

Figure 5.1: The main program for mining N-most frequent episodes.

of an episode is inversely proportionally to its size, the optimal threshold always is the support of the N-th k_{max}-episodes.

The experiment is conducted on an Sun Ultra 5_10 machine running SunOS 5.8 with 512 MB main memory. The programs are written in C. The run time measured is the total execution time of both CPU and I/O time. The run time in the experiment is the total run time of tree construction and mining. Each data points in graphs are the mean time of the several runs of the experiments. Both synthetic and real datasets are used.

5.3.1 Synthetic Data

The synthetic data are generated from the synthetic data generator [1] in [19], which is modified to generate episodes with consideration of overlapping windows. For convenience of reference, the five main parameters taken in generating data and the two datasets used in experiment are listed in Figure 5.1 and Figure 5.2.

We study the effect of N on the execution time by using different values of N from

[1]For detailed description of the modified data generator, please refer to Section 4.9.

```
Mine: To mine the $N$-most interesting episode which include a set of event types $b$
Input: Root of tree, $tree$ and a list of required thresholds, $\xi$.
Output: A frequent episode list, $F$
Procedure Mine($tree, b, \xi_{|b|+2}, ..., \xi_{c\_max}$)
{ If $tree$ contains single path only
        generate all combinations of episodes directly
  Else
        For each event type $e_i$ in header table (from top to down)
            $complete\_path = 0$; $c\_max = 0$
            For each prefix path of $e_i$
                Accumulate the total support of each event type in prefix path
                If the current path is an complete path
                    $complete\_path = 1$; $completePath\_count$ += $e.count$
                Else If ($n > c\_max$)
                    $c\_max = $ |event types in the prefix path| + $|b| + 1$
            If $complete\_path == 1$ and $completePath\_count \geq \xi_{|b|+i+1}$
                Pre-insert the episodes $b \cup e_0 \cup e_1 ... \cup e_i$ into $F$; Update $\xi_{|b|+i+1}$
            For each event type, $a$, in prefix paths
                If ($a.support \geq \xi_{|b|+2}$)
                    Insert $a \cup e_i \cup b$ into $F$; Update $\xi_{|b|+2}$
            $\xi_{min} = min(\xi_{|b|+3}, ..., \xi_{c\_max})$
            For each event type, $c$, in prefix paths
                If ($c.support \geq \xi_{min}$) then insert $c$ into freq event type list, $E$
            Construct $b \cup e_i$'s conditional tree, $tree_{b \cup e_i}$ with $E$
            If $tree_{b \cup e_i}$ not null
                Mine($tree_{b \cup e_i}, b \cup e_i, \xi_{|b|+3}, ..., \xi_{c\_max}$)
}
```

Figure 5.2: The code for mining N-most frequent episodes.

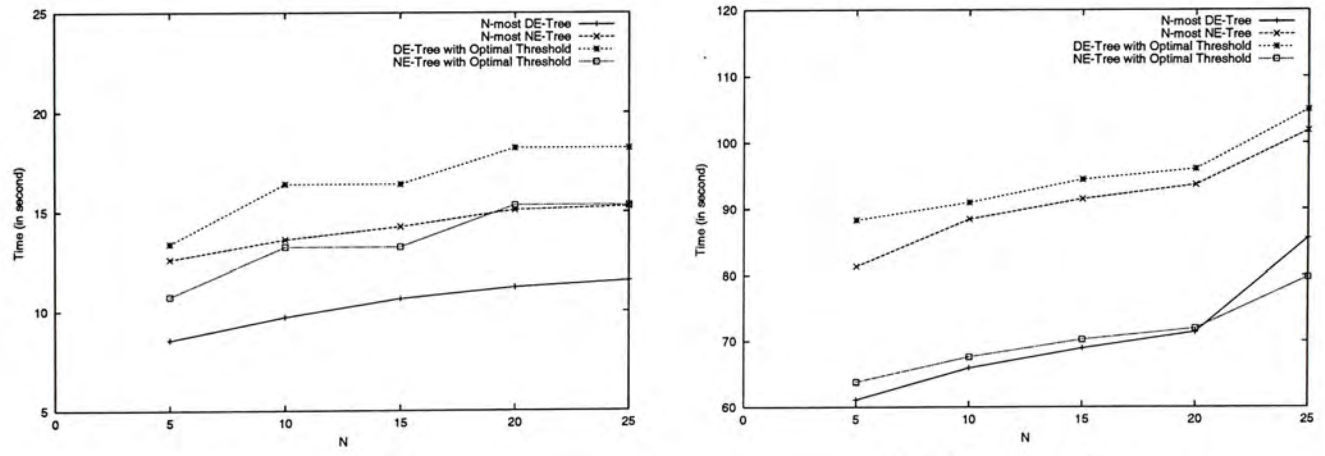
Parameter	Description	Values
$\|D\|$	Number of days	1K, 2K, 3K
$\|T\|$	Average number of event types per day	10, 20
$\|I\|$	Average size of frequent episodes	3, 5
$\|L\|$	Number of frequent episodes	1000
M	Number of event types	100 - 1000
W	Window size	2 - 10

Table 5.1: Parameter settings of the synthetic data generator.

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

Table 5.2: Two datasets with different parameter settings used in the experiment.

5 to 25 with k_{max} and window size fixed to 5 and 3 respectively. The execution time for the two datasets are shown in Figure 5.3(a) and 5.3(b). The optimal thresholds used are from 1.5% to 1.8% for D1 and from 2% to 2.34% for D2 when N decreases from 25 to 5.



(a) Dataset D1 (b) Dataset D2

Figure 5.3: Run time for different values of N with $k_{max} = 5$ and window size = 3.

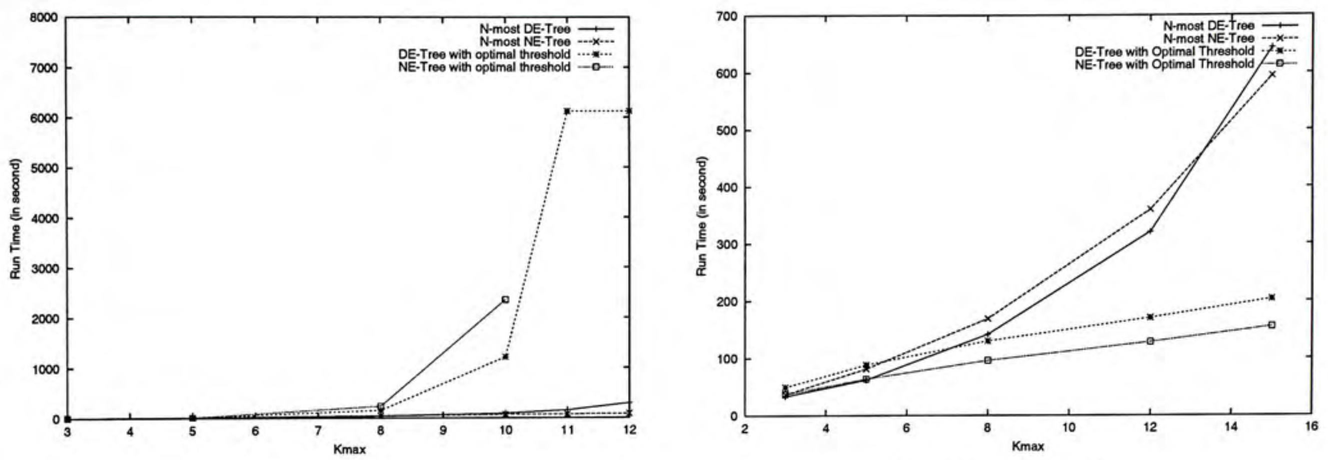
The run time increases with N because when N becomes larger, less frequent episodes are included in the resulting set of episodes. The minimum threshold, ξ_{max}, is smaller as the N increases.

The overall performance of the N-most DE-tree is the best especially when N is small. Due to the pre-evaluation of initial threshold and top-down scan of header table, the threshold improvements allow the largest possible thresholds used for each size of episode. On the other hand, in the original tree the optimal threshold, which is also the minimum threshold among the episodes mined in the results, is used in the whole process. Hence the original tree run little bit longer than the N-most tree when N is small.

However, the performance of N-most NE-tree is less satisfactory than that of the N-most DE-tree for the 'valid episode count' in the N-most NE-tree. When constructing the initial complete tree and the subsequent conditional tree, each event node is given a valid episode count. The bits in each window nodes belonging to the ancestor nodes have to be checked for the existence of zero. So extra time is spent on checking the bits in building conditional trees.

The performance of the original trees with optimal threshold is better than that of the N-most trees when N and the size of dataset are large, ie. when $N > 15$ and D2 is used. It is because the trees with optimal threshold has given a threshold which determines and removes the items the support of which less than the threshold and reduces the size of the initial tree. In addition, since the threshold given is the optimal threshold, the trees constructed are the smallest required trees. On the contrary, in the methods for mining N-most episodes, the initial tree with all items regardless of their supports is built. Therefore extra event types and extra conditional trees are included, increasing the overhead. Also when the dataset is large, the cost of constructing the initial complete tree for the N-most trees becomes significant.

The performance of the methods with different k_{max} on the datasets D1 and D2 is shown in Figure 5.4(a) and 5.4(b). N and the window size are set to 5 and 3 respectively. The optimal threshold used is from 0.5% to 5% for D1 and from 1.3% to 3.8% for D2. The run time for NE-tree with optimal threshold is too large when k_{max} is larger than 10, and it can not be measured.

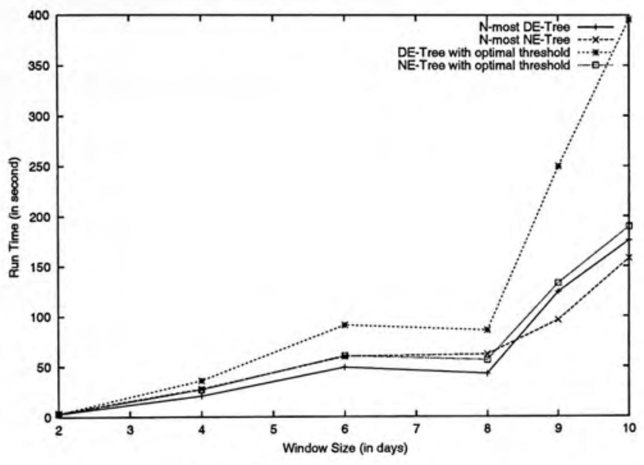


(a) Dataset D1 (b) Dataset D2

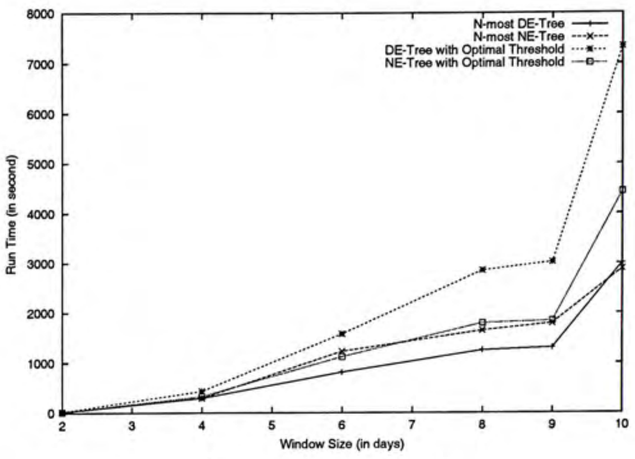
Figure 5.4: Run time for different values of k_{max} with $N = 5$ and window size $= 3$.

The run time increases with k_{max}. Since the size of episode is inversely proportional to the support, the minimum threshold of the N-most trees is always equal to the N-th support of the k_{max}-episodes. The minimum threshold and the optimal threshold is getting smaller when k_{max} is large.

We also investigate the effect of the different window sizes on the execution time. We have set the window size from 2 to 10 with both N and k_{max} kept to 5. The results are shown in Figure 5.5(a) and 5.5(b). The optimal thresholds for D1 and D2 are from 19.6% to 1% and from 1.74% to 21.87% respectively. The run time rises exponentially with the window size.



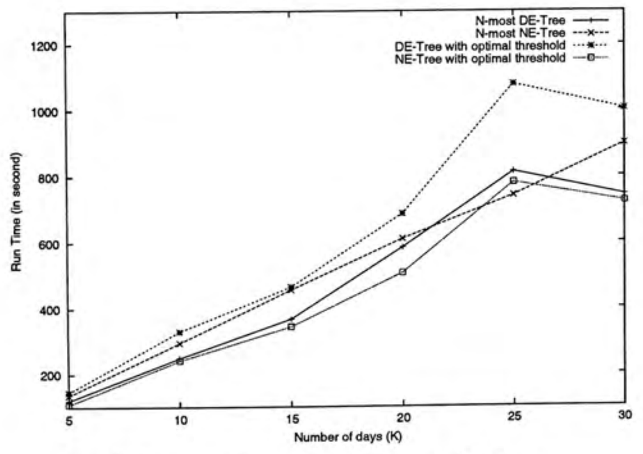
(a) Dataset D1



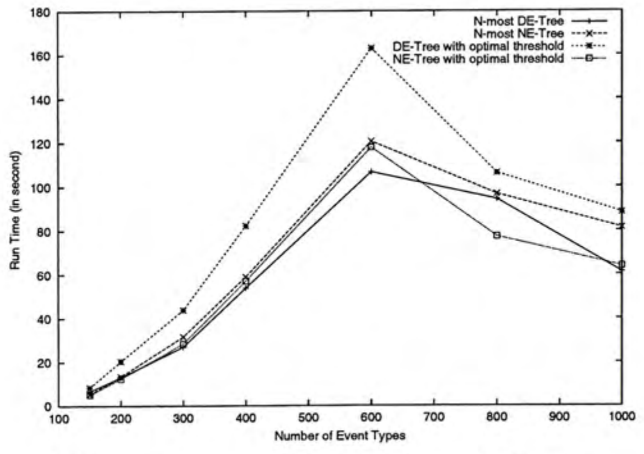
(b) Dataset D2

Figure 5.5: Run time for different values of N with $k_{max} = 5$ and window size $= 3$.

We also evaluate the effect of the different number of days, D, on the run time. Experiment on dataset D2 is conducted. The N, window size and k_{max} are set to 5, 3 and 5 respectively. The optimal threshold is from 2.225% to 2.36%. The run time increases with the number of days linearly as shown in Figure 5.6(a).



(a) varying number of days



(b) varying number of event types

Figure 5.6: Run time for dataset D2 with $N = 5$, window size $= 3$ and $k_{max} = 3$.

The performance of the methods under different number of event types are compared and shown in Figure 5.6(b). The values of N, k_{max} and window size are set to

5, 5 and 3 respectively. The optimal threshold is from 2.34% to 35.1%.

The run time increases with the number of event types, which is different from the decreasing run time for the original trees in Section 4.9. Since the value of N is given in N-most trees and the initial thresholds are deduced from N, the thresholds are dynamically adjusted for the dataset. Thus even though the frequencies of the smaller number of event types are higher, the higher initial thresholds are estimated. With the smaller number of event types and higher thresholds, less run time is resulted.

5.3.2 Real Data

We also use the real dataset produced from the events extraction from newspapers in Chapter 3 to measure the performance on varies N, k_{max} and window sizes as shown in Figure 5.7 to 5.9.

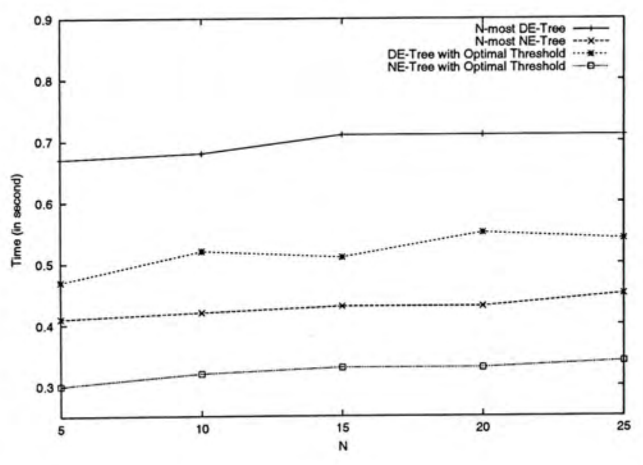


Figure 5.7: Run time for different values of N on real dataset with k_{max} = 5 and window size = 3.

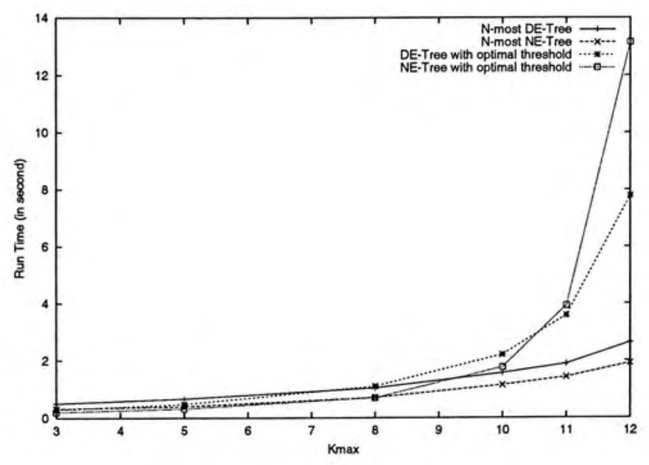


Figure 5.8: Run time for different values of k_{max} on real dataset with $N = 5$ and window size = 3.

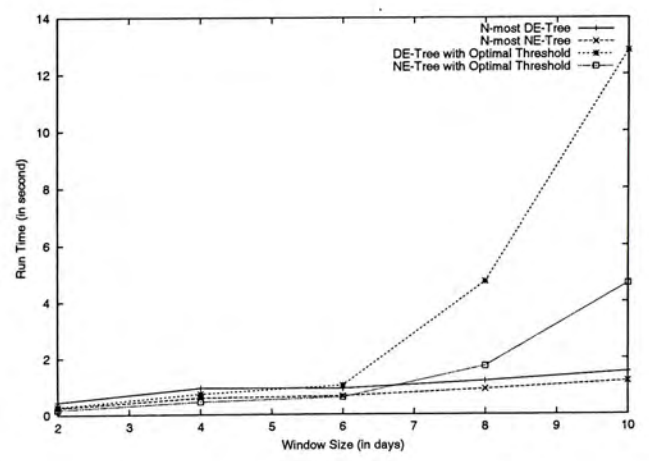


Figure 5.9: Run time for different window sizes on real dataset with $N = 5$ and $k_{max} = 5$.

The two N-most trees perform better than the original trees with optimal threshold under large window sizes. Since in real dataset the large sizes of episodes are more frequent, and the optimal threshold used in this case are the minimum support among the 2-episodes mined from the N-most trees. The pruning power is less in building the subsequent conditional trees with the original trees with optimal threshold.

5.4 Conclusion

In this chapter, we have proposed a method to mine the N-most interesting k-episodes. Experiments show that the proposed method is comparable to the original trees introduced in Chapter 4 mining episodes with an optimal threshold given. Although the performance of the original trees are better than the N-most tree under certain conditions, it is due to the the optimal thresholds, which are given by the mined result from N-most tree. In addition, the N-most trees provides a convenient way that the user just needs to specify the number of the most frequent k-episodes desired, which is more practical and direct. Therefore, the method of mining N-most frequent episodes has eliminated the difficulty and the troublesome of adjusting the threshold in order to obtain the suitable number of episodes.

Chapter 6

Mining Frequent Episodes with Event Constraints

In the previous chapter, the method of mining N-most interesting episodes is presented. Although it has eliminated the problem of deciding the minimum support threshold, the result mined still may not be what we desire. In reality, we usually want to obtain those episodes with certain event types. In addition, the previous methods also fail to get the episodes with low support in a reasonable time. In this chapter, we introduce a method of mining of N-most interesting episode with event constraints. The experiment shows that the episodes with specified event types regardless of their support can be mined in an acceptable time.

6.1 Introduction

In mining N-most interesting episodes, the event types in the mined episodes are usually also the frequent event types. However, in many cases, the frequencies of the event types are not uniform. For example, in the real dataset produced in Chapter 3, the number of the newspaper event types is much less than the number of the stock event types. But the episodes related to the newspaper event types are our main target. When we are interested in the episodes associated with the less frequent episodes, the N-most method cannot find the episodes containing the event types unless larger values of N are set. But this trial-and-error way of finding the suitable N is not the objective of the N-most method. Thus it will be more efficient to specify the event types that will be desirable in the results.

The similar problem of mining with event constraints, which is called item constraint in general, is studied in [31] and [30]. [31] proposes the support-constraint which applies to the Apriori-gen algorithm to mine the itemset under non-uniform minimum support. The support-constraint is the minimum support threshold that the itemset containing the specified item constraints should meet. The items is ordered so that the performance is optimized.

[30] proposes single FP-tree and double FP-tree approaches to discover interesting itemsets with item constraints. It mines the itemset specified by the 'constraints and bins' under closed and open interpretation defined in [31]. For the single FP-tree, it modifies its own N-most tree which is based on the FP-tree [23] and adjusts the support thresholds by considering the N-th supports of all itemsets containing the item constraints.

In the double FP-tree, it uses a FP-tree to store the set of constraints to reduce the cost of memory size and the run time on matching the itemset with constraint.

The method we propose to mine the frequent episodes with event constraints is based on the single FP-tree in [30] with some modifications.

6.2 Method

We introduce a method of mining N-most interesting episodes with event constraints. An **event constraint**, C, is a set of events that should appear in each of the episodes mined in final results.

The episodes in the final results consists of:

1. **N_i-most interesting k-episode with event constraints**: For each given event constraint, C_i, with N_i, $0 \leq i <$ number of constraints, the N_i most

frequent k-episodes, $2 \leq k \leq k_{max}$, containing C_i will be mined.

2. **N-most interesting k-episodes**: For each size of episode $\leq k_{max}$, the top N frequent episodes from the set of episodes, excluding the episodes containing any E_i, will be mined.

For example, given 2 event constraints, C_0 = {Government announces deficit, Hang Seng Index ups}, C_1 = {Government announces deficit, Hang Seng Index downs}, $N_0 = N_1 = 2$, $N = 1$ and $k_{max} = 3$, the possible episodes in final results are shown in Table 6.1.

Event Constraint	k	Possible Episodes Mined
C_0	2	1. Government announces deficit, Hang Seng Index ups
	3	1. Government announces deficit, Hang Seng Index ups, China Mobile ups 2. Government announces deficit, Hang Seng Index ups, HSBC ups
C_1	2	1. Government announces deficit, Hang Seng Index downs
	3	1. Government announces deficit, Hang Seng Index downs, HK & China Gas flats 2. Government announces deficit, Hang Seng Index downs, Hang Seng Bank ups
-	2	1. Land auction, Cheung Kong ups
	3	1. US interest rate increases, Hang Seng Index ups, Dow Jones ups

Table 6.1: The possible episodes in the final results.

Slightly different from [30], we also include the N-most interesting k-episodes for completeness of results. Since if the results only contains the N_i-most interesting k-episode with event constraints, we cannot know the N-most interesting k-episodes, which also gives the important episodes. Although we can use the pure N-most event trees, the N-most interesting k-episodes obtained from this method are excluded those sets of event types in constraints and may be different from the results from the pure N-most event trees.

We now describe a method developed from the event trees in Chapter 5. At the beginning, an initial complete tree is built with threshold set to 1. In mining phase, apart from the original individual support thresholds, ξ_k for each size, k, of episodes, each event constraint, C_i, is assigned to its own set of support threshold, θ_{ik} for each size, k, of episodes. The individual support thresholds, θ_{ik}, is equal to the N_i-th highest support of the corresponding k-episodes for C_i currently mined. Similarly, the individual threshold, ξ_k, is the N-th highest support of k-episodes, which do not contain any set of event types in the constraints, mined so far.

Thus when determining if an k-episode, e, is frequent and can be inserted into the result episode list, the support of e is checked and compared. The support threshold compared for e is as follows:

$$\text{Support threshold compared with} \begin{cases} \theta_{ik} & \text{if } e \text{ contains all events in } C_i \\ \xi_k & \text{otherwise} \end{cases}$$

If the support meets the corresponding individual support threshold, e is added to the corresponding result list and the individual threshold is updated.

The minimum threshold ξ_{min} that used to select the event types for constructing the next conditional event tree is

$$\xi_{min} = min(\text{all possible } \xi_k \text{ and } \theta_{ik})$$

where |base events| + 3 $\leq k \leq$ |base events| + |longest prefix path| and where |longest prefix path| is smaller than the number of ancestor event types in the header table.

The techniques for threshold improvement, which are top-down header table scan and pre-insertion of episodes, introduced in the previous chapter are also applied.

6.3 Experimental Results

We measure the performance of the method applied to the two event trees. The experiment is conducted on an Sun Ultra 5_10 machine running SunOS 5.8 with

512MB Main Memory. The programs are written in C. Both synthetic and real datasets are used.

6.3.1 Synthetic Data

The synthetic data are generated by the modified synthetic data generator[1] from [19]. Two datasets, D1 and D2, with the parameter settings listed in Table 6.2 are used.

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

Table 6.2: Two datasets with different parameter settings used in the experiment.

The event constraints for the synthetic datasets are created randomly. The number of event types in the constraints varies from 1 to 5. The total number of constraints of D1 and D2 are 158 and 341 respectively. The sum of the numbers of event types in all constraints are equal to the number of event types in the dataset that the event types are unique in all constraints created.

The N_i for each constraint C_i is determined by the equation

$$N_i = \frac{\alpha}{|C_i|} \sum_{j=0}^{|C_i|-1} s_{ij}$$

where s_{ij} is the support of the event type e_j in C_i and α is the factor of the average number of episodes to be taken as the top N_i interesting episodes containing event constraints. The properties of the constraints generated are summarized in Table 6.3. Except N_i, α and the maximum number of event types in constraints, all properties are randomly determined.

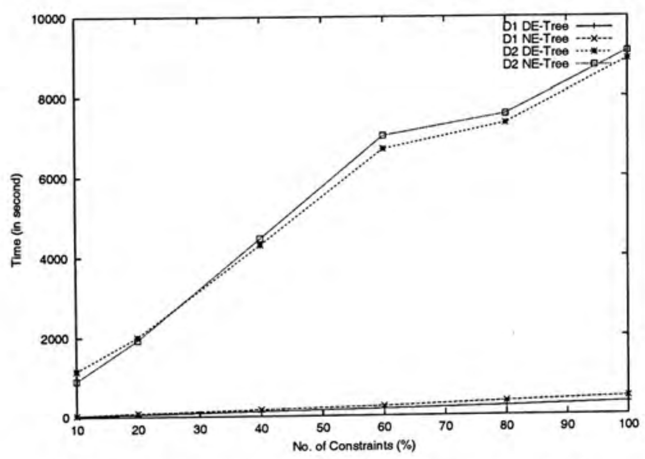
We evaluate the performance on various numbers of constraints by randomly choosing 10% - 100% of the whole set of constraints. The run time of the two event

[1]For detailed description of the modified data generator, please refer to Section 4.9.

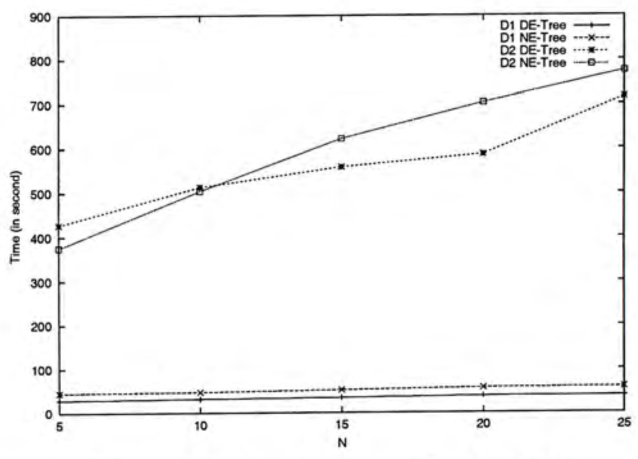
Dataset	No. of Constraints	Max Size	Min Size	Avg Size	Max N	Min N	Avg N	α
D1	158	5	1	3	24	1	9	0.1
D2	341	5	1	3	17	1	7	0.1

Table 6.3: The properties of the constraints created for two datasets.

trees are shown in Figure 6.1(a). Both N and k_{max} are set to 5 and the window size is 3 days. The execution time increases with the number of constraints. As more constraints are set, more time is spent on checking and matching the episodes with the event types in the constraints. The minimum thresholds are also lower because more thresholds for constraint are included in calculating the minimum threshold. The sizes of the conditional event trees are larger and more time needed for mining.



(a) varying number of constraints with $N = 5$

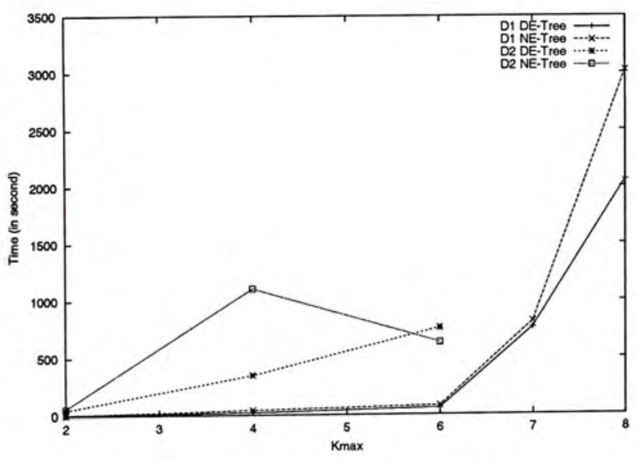


(b) varying number of N with 10% of the total constraints

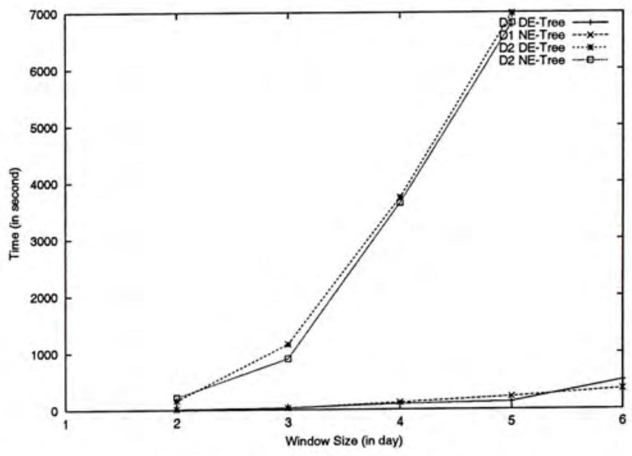
Figure 6.1: Run time for synthetic datasets D1 and D2 with window size = 3 and $k_{max} = 5$.

The performance on various values of N is also investigated and shown in Figure 6.1(b). The N and N_i for all constraints C_i are the same value. 10% of the total constraints are used. k_{max} and window size are 5 and 3 days respectively. It shows the execution time rises with N. As N increases, the thresholds are smaller. The execution time is larger for less events pruned in forming conditional event trees.

We also studied the effect of performance on different k_{max} which is shown in 6.2(a). The N and window size are kept to 5 and 3 days respectively. 10% of the total constraints are used. The execution time increases exponentially with k_{max}. The run time for the two event trees on D2 when k_{max} is above 6 exceeds 18000 seconds which is too large to be measured.



(a) varying number of k_{max} with window size = 3



(b) varying window sizes with $N = 5$

Figure 6.2: Run time for synthetic datasets D1 and D2 with 10% of the total constraints and $N = 5$.

The performance on the various window sizes is also tested and is shown in Figure 6.2(b). 10% of the whole set of constraints are selected, and both N and k_{max} are set to 5. The run time increases exponentially with the window size. When the window size is 6 days, the run time for the two event trees using D2 are greater than 16800 seconds and cannot be measured.

The average sizes of the two event trees residing in memory are less than 100MB. In the extreme case, when the run time is long, the size will reach 200MB. In general, it occupies more memory space than the methods in the previous two chapters. It is because the thresholds are smaller and more event types are reserved. Also, extra memory storage for the k-episode results for each constraint has to be allocated.

6.3.2 Real Data

The real dataset used is the event database generated from Chapter 3. Similar to the synthetic data, we create a set of constraints randomly. With the maximum size of constraint and α as 5 and 0.1 respectively, total 41 event constraints are generated. The minimum size of constraint is 1. The values of N range from 1 to 50 and its average is 18.

The performance of two event trees on different numbers of constraints is shown in Figure 6.3. Both N and k_{max} are set to 5, while the window size is set to 3 days. The execution time rises with the number of constraints.

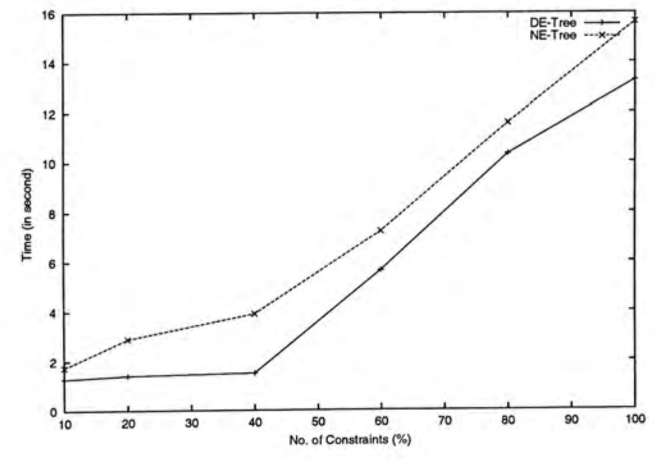
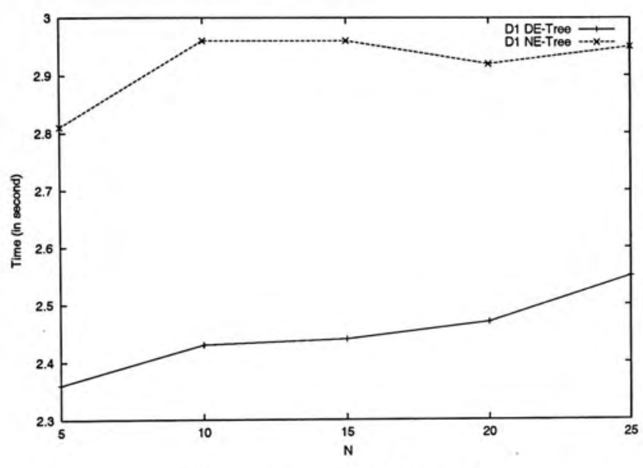
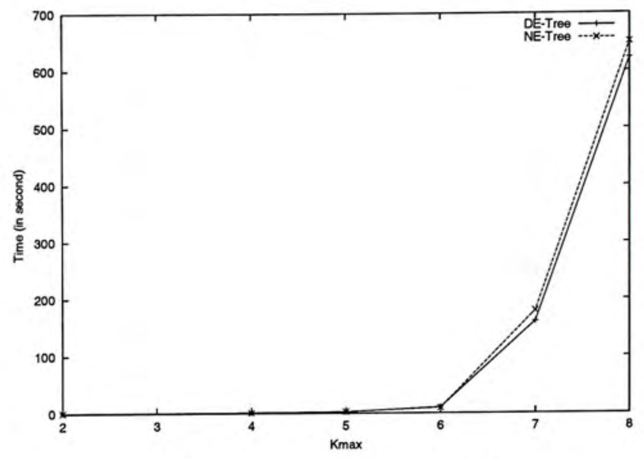


Figure 6.3: Run time for real dataset for different numbers of constraints with $N = 5$, window size = 3 and $k_{max} = 5$.

We also studied the effect under various N, k_{max} and window sizes, the performance of which is shown in Figure 6.4 to 6.5 respectively. 20% of the whole set of constraints are selected, with both N and k_{max} as 5 and window size as 3 days by default.



(a) varying N with $k_{max} = 5$



(b) varying k_{max} with $N = 5$

Figure 6.4: Run time for real dataset with 20% of the total constraints and window size = 3.

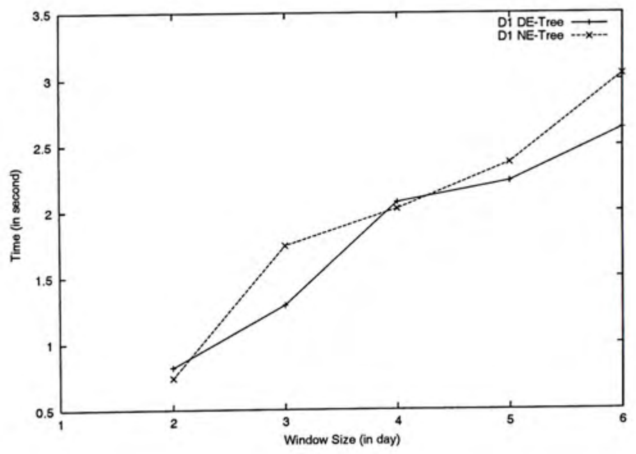


Figure 6.5: Run time for real dataset for different window sizes with 20% of the total constraints, $N = 5$ and $k_{max} = 5$.

Episodes mined

In this section, we set the constraints manually by specifying the event types we are interested and investigate the episodes associated.

We focus on the event types of news release. By setting k_{max} to 4 and the window size to 3 days, the episodes mined is listed in Table 6.4.

Event Constraint		Additional Events in Episode	Support	# of Episodes Mined
Government announces deficit	Hang Seng Index ups	-	11	3
		China Mobile ups	13	
	Hang Seng Index downs	-	12	2
		HK & China Gas flats	18	
Report released	Hang Seng Index ups	-	10	30
		CLP Holding downs	8	
	Hang Seng Index downs	-	12	2
		Cheung Kong downs	10	
Land auction		Cheung Kong ups	26	25
		SHK Properties ups	25	
		Cheung Kong ups, Hutchison ups	26	
New Plan from China Unicom		China Unicom ups	3	2
		PCCW flats	2	
Bad news from China Mobile		China Unicom ups	3	15
		PCCW ups	2	
Cheung Kong buys land with low price		Cheung Kong ups	2	26
		Cheung Kong flats	5	

Table 6.4: The episodes mined for the constraints of news events.

6.4 Conclusion

In this chapter, a method of mining the N-most frequent k-episodes with event constraints is introduced. It complements the methods in the previous two chapters that episodes associated with the specific event types can be mined even if their supports are low, without setting a low threshold and over-generating a long list of episodes.

There is some room for improvement for the performance of the mining method proposed. One possible solution is to construct the conditional trees of the specific event types for each constraint to reduce the matching cost in building the conditional trees. Thus in mining the N-most interesting episodes, higher support thresholds can be archived because there is no need to consider the thresholds of event constraints.

Since the tree structures are not designed for retrieving the episodes containing the specific event types, the trees can be modified to suit the application. For example, the double FP-tree in [30], which has the basic structure as original FP-tree, is proposed to mine the itemset with item constraint. Its experimental results shows the performance outperform the original FP-tree.

However, the run time is still acceptable for a non-real-time system. The method is also useful for the convenience it brings.

Chapter 7

Conclusion

Information extraction is the process of unrestricted text into a systematic data and is considered as the preliminary step for preparing data for further analysis. We implemented the IE systems and performed the subsequent data analysis with data mining techniques. We applied the IE system and data mining to discover the frequent episodes in the stock market.

In the first part, we built an automatic summarization system for Chinese documents. It extracts the necessary sentences from the Chinese financial news with genetic algorithm. It considers the frequencies of keywords and the locations of sentences, the proximities and distances between summary sentences in forming the summary.

On the other hand, we discovered the frequent episodes involving stock movement and news releases from stock companies. We constructed a real database with the news events extracted from the Chinese financial newspapers. We used two new mining methods to discover the frequent episodes with a given threshold, which are based on two corresponding tree structures. Although the methods show the effectiveness on performance, they need a pre-assigned threshold which impose a difficulty for user who has little knowledge about the database. We enhanced the system and proposed a method which mines the frequent episodes with the number of the most frequent episodes assigned rather the support threshold. We also introduced the method of mining with event constraints which provides a direct way to obtain the user-specific episodes. We can also discover those episodes with small supports with the event constraint method.

With the three effective methods, we can efficiently handle the task of mining frequent episodes under databases with different frequencies distribution and different user requirements. In the future, the mined frequent episodes can be analysed for stock prediction.

Appendix A

Test Cases

Here we show text 1 and text 2 used in the experiment in Chapter 2. The sentences selected as the summary sentences are shown in italic.

A.1 Text 1

香港醫療架構從來是黑箱作業，近期的「手機醫生」事件，活生生就是醫醫相衛的例子。

「手機醫生」董曉明，在為病人進行切除大腸息肉手術期間，兩次使用免提裝置傾談電話，手術後當晚，病人大腸爆裂，需再進行緊急手術。 *個案經醫委會聆訊後，裁定病人事後的併發症與醫生的傾談手機行為沒有關係，而「手機醫生」亦被裁定沒有行為失當。*醫委會的裁決大抵憑藉兩個理由：醫生在手術進行中講話並非不尋常；該醫生亦非刻意在手術期間談電話。

醫委會的判詞含糊不清，大有玩弄文字之嫌。*判詞指「醫生在手術進行中講話並非不尋常」，然而，該醫生並非在手術中談話那麼簡單，而是在手術期間使用手提電話談話。*純粹談話和傾談手機根本不可混為一談，兩者性質不同，談話者的注意力也不同。醫生在手術期間需要和其他醫護人員合作，期間當然少不免有交談的機會，例如向護士要求幫助、更換手術用具等，或與其他醫護人員交換意見，以求達至更準確精密的手術。然而，在手術期間使用手提電話卻不可同一而喻。首先，手術房中有各種醫療儀器，使用手提時放出的電波、

輻射，手機會影響醫療儀器的正常運作。所有公立醫院都有張貼海報，提醒市民在病房附近必須關掉手提，醫生帶手機進手術室，明顯是知法犯法。

傾談電話時，人的注意力很自然地會轉移向電話的另一端，分散對眼前事物的注意力。巴士公司禁止員工行車時和乘客談話，也是同一道理。在道路上，巴士司機掌握著車上眾多乘客的生命安全，自然不可輕率行事。同樣道理，在手術室中，醫生手握手術刀，握病人的生死存亡於一線，醫生更須聚精會神，心無旁騖。*在手術期間傾談電話，非但是對病人的不尊重，同時亦是對自己身分的不尊重，更削弱病人對醫生的信任。*

醫委會認為，「手機醫生」出於一時大意，忘記在進入手術室前關掉手提，接聽電話是源於無奈，當他聽了電話後，已盡快中止談話，這次事件純粹是無心之失。 事實真的如此嗎？「手機醫生」當時是使用免提裝置傾談電話，換句話說，除了一時大意放進口袋裏的手提電話，醫生的耳洞裏還塞著迷你聽筒，這樣礙眼又不舒服的裝置，說是出於大意，可信性有多少？手機響了，醫生大可選擇立即關閉電話，他有拒絕接聽的可能，但他沒有做到。該醫生在手術期間，一共聽了兩次電話，若是他是在無奈下接聽了第一次電話，那麼第一次談話結束後，為甚麼他沒有關閉電話，而是一錯再錯地接聽了第二次電話？若說一次過失是無心之失，那麼第二次錯誤無論如何也不能算是無心之失了。最可恨的是，該醫生竟然在電話大談汽車買賣事宜，完全與手術無關，談話時間更長達十多分鐘。這樣缺德的「無心之失」，難為醫委會還可以處處維護，顛倒是非來包容，不愧為「公平公正」的醫務委員會。

香港的醫療體系從來是醫醫相衛，歷來醫療失誤的新聞不絕於耳，但何曾聽聞過醫委會對失誤醫生有重大懲罰？ 香港醫生，九成出自港大醫學院，師兄弟從來是站在同一線，互相偏袒護短。醫生失誤，醫院首先出來擋駕，封鎖一切資料、保密一切紀錄。同行醫生拒絕做証人，拒絕作供，沒有人証、物証，所有起訴都難於成功，既得利益者壟斷一切監察機制，包庇護短，醫委會本身不過是徒然具名的幌子。

「手機醫生」事件暴露出香港醫療監察機制的不健全，港府應盡快解散現有的醫委會，將之重組，加重非業內代表的比例，並注重委員會的客觀性，在委任各委員時，著重各人的公信力、品格、誠信，重拾公眾對醫委會的信任。同

時，由政府撥出公帑，成立專責調查委員會，聘請外國專家對失誤醫生做出全面調查，杜絕業內醫醫相衛，包庇偏私。若証實醫生專業失當，致令病人蒙受不必要的損失，可向之追究刑事責任。港府亦可開放市場，批准國內專家級醫生來港執業。香港醫生面對來港國內醫務專才的競爭，自然有龐大動力推動自己提昇水準，提高服務水平。

「手機醫生」的缺德行為不容醫委會砌詞狡辯，港府應及早改組醫委會，重拾公眾對醫學界的信任。

A.2 Text 2

銀行界將於七月正式取消利率協議。*此舉不僅對銀行業經營環境造成影響，對銀行員工、存戶及整體經濟亦會帶來影響。*從宏觀經濟來看，取消利率協議會加強銀行間的互相競爭，催化經濟轉型，更會擴闊市民的收入差距，因而加劇本港的貧富懸殊問題。

*取消利率協議，銀行經營模式改變，銀行員工首當其衝。*利率協議取消後，預期利息差距將會收窄到一至兩個百分點，從事傳統貸款及存款業務的邊際利潤會大大下降，銀行會推廣各式各樣的投資產品，來提升自己的邊際利潤。因此，傳統的分行業務會由電子理財取代，而大部分的分行會改為投資理財中心，基層職員將要充當推銷投資產品的財務顧問。

為配合發展投資理財的銀行服務，銀行需要聘請更多高學歷人士，這是香港轉向知識型經濟的必然道路。可是，現在的基層員工，尤其是中年員工，是否有足夠的知識和能力擔任理財顧問一職實屬疑問。雖然銀行願意增撥資源，加強培訓，但大部分受影響員工在短期培訓後，未必能夠適應全新的知識型工作崗位。在預定時間內未能適應新工作者，將會難逃被撤職的厄運，而他們的工作崗位，將會由七月開始輸入的內地財經專才所取代。

銀行業在業務經營模式改變下，自然不能完全吸納這批受影響的基層員工，如果從事銀行多年的基層員工，沒有其他工作經驗或技能，成功轉到其他行業的可能性將會很低，這批員工只會淪為長期失業大軍的一員，直接推高本港的失業率。

*從表面看來，在利息分層制度下，小存戶所損失的只是少量利息收入，不過在銀行業大量推出投資產品下，擁有資產者更加容易賺取較高的投資回報。*相反，小存戶不僅只能賺取比以前更低的利息，而且亦沒有其他資產可作投資用途，加上銀行業傾向對小額存戶收取服務費用，縱使費用只是數十元的小數目，但對於低下階層的生活質素，已經構成很大的影響，所以取消利率協議後，令貧者愈貧，富者愈富，社會的貧富收入差距擴大，有礙整體的經濟穩定。

銀行電子化影響低下層銀行業務日趨電子化，現在有不少上了年紀的公公婆婆不懂得如何操作自動櫃員機，加上在自動櫃員機提款的人比用自動櫃員機存款的人多出數倍，大多數人慣用櫃位服務來處理存款。所以小額存戶不一定能夠適應自動櫃員機的銀行服務形式。如果將來的銀行業務由網上理財取代自動櫃員機，受影響的人數將會大幅提升。

就以新加坡為例，該國計劃於〇八年開始推行電子貨幣，取代傳統的紙幣和硬幣，以期成為亞太地區的高科技中心。若香港的發展方向與新加坡相同，在不久的將來市民便要懂得在網上提款，和懂得透過已裝置聰明晶片的手機、手提電腦、腕表等來處理「過數」。在電子貨幣普及化的未來，很多不懂得運用互聯網的人士，會失去使用傳統銀行提款和存款服務的能力，估計低學歷和低技術的基層人士所受的影響會最大。

Bibliography

[1] D. Marcu, From discourse structures to text summaries, in *Proceedings of the ACL'97/EACL'97 Workshop on Intelligent Scalable Text Summarization*, pages 82–88, 1997.

[2] M. Mitra, A. Singhal, and C. Buckley, Automatic text summarization by paragraph extraction, in *Proceedings of the ACL'97/EACL'97 Workshop on Intelligent Scalable Text Summarization*, pages 39–46, 1997.

[3] R. Barzilay and M. Elhadad, Using lexical chains for text summarization, in *Proceedings of the ACL'97/EACL'97 Workshop on Intelligent Scalable Text Summarization*, pages 10–17, 1997.

[4] S. Teufel and M. Moens, Sentence extraction as a classification task, in *Intelligent Scalable Text Summarization Proceeding of a Workshop ACL'97*, pages 58–65, 1997.

[5] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke, Seeing the whole in parts: Text summarization for web browsing on handheld devices, in *The 10th International WWW Conference (WWW10)*, pages 652–662, 2001.

[6] S. Teufel and M. Moens, Argumentative classification of extracted sentences as a first step towards flexible abstracting, in *Advances in Automatic Text Summarization. The MIT Press*, pages 155–171, 1999.

[7] P. D. Turney, Learning algorithms for keyphrase extraction, in *Information Retrieval*, pages 303–336, 2000.

[8] A. Munoz, Creating term associations using a hierarchical art architecture, in *C.v.d. Malsburg and W.v. Seelen, editors, International Conference on Artificial Neural Networks, Lecture Notes in Computer Science*, pages 171–177, 1996.

[9] E. Frank, G. W. Paynter, I. H. Witten, and C. Gutwin, Domain-specific keyphrase extraction, in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 668–673, 1999.

[10] J. Goldstein, M. Kantrowitz, V. O. Mittal, and J. G. Carbonell, Summarizing text documents: Sentence selection and evaluation metrics, in *SIGIR 1999*, pages 121–128, 1999.

[11] J. H., R. Barzilay, K. McKeown, and M. Elhadad, Summarization evaluation methods: Experiments and analysis, in *Intelligent Text Summarization. AAAI Press*, pages 51–59, 1998.

[12] A. Ng and K.H.Lee, Event extraction from chinese financial news, in *International Conference on Chinese Language Computing (ICCLC)*, 2002.

[13] D. Freitag, Information extraction from html: application of a general machine learning approach, in *AAAI-98*, pages 517–523, 1998.

[14] M. E. Califf and R. J. Mooney, Relational learning of pattern-match rules for information extraction, in *16th National Conference on Artificial Intelligence*, pages 9–15, 1999.

[15] H. Ahonen, O. Heinonen, M. Klemettinen, and I. Verkamo, Applying data mining techniques for descriptive phrase extraction in digital document collections, in *Digital Libraries'98, Santa Barbara, California, USA*, pages 2–11, 1998.

[16] U. Y. Nahm and R. J. Mooney, Using information extraction to aid the discovery of prediction rules from text, in *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, pages 51–58, 2000.

[17] S. D'Alessio and R. S. Keitha Murray, The effect of using hierarchical classifiers in text categorization, in *Proceeding of RIAO-00, 6th International Conference*, pages 302–313, 2000.

[18] H. Mannila and H. Toivonen, Discovering generalized episodes using minimal occurrences, in *2nd International Conf. On Knowledge Discovery and Data Mining*, pages 146–151, 1996.

[19] R. Agrawal and R. Srikant, Mining sequential patterns, in *11th International Conf. On Data Engineering*, pages 3–14, 1995.

[20] H. Mannila, H. Toivonen, and A. I. Verkamo, Discovering frequent episodes in sequences, in *1st international Conf. On Knowledge Discovery and Data Mining*, pages 210–215, 1995.

[21] K. Hatonen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen, Knowledge dicovery from telecommunication network alarm databases, in *Proceedings of the 12th IEEE International Conference on Data Engineering*, pages 115–122, 1996.

[22] P. S. Kam and A. W. C. Fu, Discovering temporal patterns for interval-based events, in *Proc. Second International Conference on Data Warehousing and Knowledge Discovery*, pages 317–326, 2000.

[23] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation, in *SIGMOD*, pages 1–12, 2000.

[24] M. Chen, J. Park, and P. Yu, Efficient data mining for path traversal patterns, in *IEEE Transactions on Knowledge and Data Engineering*, pages 209–221, 1998.

[25] H. Lu, J. Han, and L. Feng, Stock movement prediction and n-dimensional inter-transaction association rules, in *Proc. of SIGMOD workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98)*, pages 1–7, 1998.

[26] H. F. Pak Chung Wong, Wendy Cowley and E. Jurrus, Visualizing sequential patterns for text mining, in *Proceedings IEEE Information Visualization*, pages 105–113, 2000.

[27] J. Pei et al., Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, in *Proceedings of the 12th IEEE International Conference on Data Engineering*, pages 215–226, 2001.

[28] R. Agrawal and R. Srikant, *Mining Sequential Patterns*, Research Report RJ 9910, IBM Research Report RJ 9910, 1994.

[29] A. Fu, R. Kwong, and J. Tang, Mining n most interesting itemsets, in *12th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pages 59–67, 2000.

[30] Y. L. Cheung, *Techniques in Data Mining: Decision Trees Classification and Constraint-based Itemsets Mining*, M.Phil. Thesis, The Chinese University of Hong Kong, 2001.

[31] K. Wang, Y. He, and J. Han, Mining frequent itemsets using support constraints, in *The VLDB Journal*, pages 43–52, 2000.