A STUDY OF FIRST LOCAL MAXIMUM OF CONFIDENCE IN MINING SEQUENTIAL PATTERNS

By

CHUANWU XIONG
Master of Science in Geology
Nanjing University
Jiangsu Province, P. R. China
1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2007

A STUDY OF FIRST LOCAL MAXIMUM OF CONFIDENCE IN MINING SEQUENTIAL PATTERNS

Thesis Approved:

Dr. H. K. Dai

Thesis Adviser

Dr. John P. Chandler

Dr. G. E. Hedrick

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to gratefully and sincerely thank my advisor, Dr. H. K. Dai, for his consistent guidance, and commitment. His sharp thinking, broad knowledge, and genuine words of encouragement will never be forgotten. I also want to express my sincere gratitude to my thesis committee members Dr. John P. Chandler, and Dr G. E. Hedrick for their suggestions, comments, and support.

Last but not the least, thanks to my wife, son, and daughter! I could not finish this thesis without their understanding and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

MOTIVATION

Data mining is also called knowledge discovery in databases. It is a process that extracts interesting information from large data repositories. The information may be some unknown patterns hidden in the data, or can be used to predict trends, behaviors in the future, or help make business decisions. With the trend of global digitalization, commercial or research organizations and governmental agencies are collecting, storing and analyzing vast amounts of data, which could be gigabyte-sized, terabyte-sized or even petabyte-sized. The need to discover valuable information from the massive data fueled the remarkable development of various data mining algorithms and applications in recent years.

The tools used in data mining include but are not limited to statistical analysis [16], mathematical modeling [19], and machine learning techniques [17].

Data mining is being applied in many domains:

- Data association: Discovering how one object or event is connected with others [5].
- Data classification: Identifying what is a good way to organize the data [20].
- Data clustering: Finding similarity among the data objects and grouping them by similar patterns [15].
- Forecasting: Discovering patterns from the available data with an aim to predict

future probabilities and trends [9].

- Sequential data mining: Searching for reoccurring patterns in sequential data and the relationship among these patterns. Sequential data are composed of ordered items or events, such as a customer's purchasing history during a long time. The focus of this study falls in this domain.

The problem of mining sequential data was introduced by R. Agrawal and R. Srikant when they were studying customers' purchasing patterns from a large database of customer transactions [1]. One of the important research questions in the sequential data mining is how to find all frequent episodes and their association rules from a large sequential dataset efficiently. A frequent episode is a collection of ordered events whose support is greater than the user-defined threshold in a research, or the number of occurrences of the episode in the sequential data must be no less than a specific value. An association rule is measured by the confidence of two episodes. It answers questions of how often episode *B* will occur when episode *A* happens in the same sequential dataset. The confidence of two episodes is the ratio of the occurrences of the two episodes. The discovery of frequent episodes and the episode rules allows better analysis and monitoring of events of interest, for example, detecting or predicting security issues in networks [12, 21, 22], user behavior prediction and fraud detection in financial systems [4], crime pattern research and helping the process of solving crime in communities [7], early detection of disease outbreaks [18], and computer forensic investigation [2].

WinEPI and *MinEPI* are two algorithms that were introduced for investigating association rules among the frequent episodes in a sequential dataset. *WinEPI* counts the occurrences of the episodes in a sliding window along the sequential data [10, 12]. The

MinEPI algorithm is based on the concept of minimal occurrence that avoids repeated counting of one episode in different windows in the *WinEPI* algorithm [11, 12]. Both the *WinEPI* and *MinEPI* algorithms need input a value of window width. The *WinEPI* algorithm retrieves the association rules of the frequent episodes with the window width input by the user. The window size in the algorithm has to be decided before each run of the algorithm. In the *MinEPI* algorithm, the input size is the maximal windows size.

N. Meger and C. Rigotti improved the *WinEPI* and *MinEPI* algorithms and introduced the *WinMiner* algorithm [13]. The new algorithm extracts the association rules of frequent episodes of different sizes in a sequential dataset and also looks for the first local maximum of the confidence regarding the window sizes. But the algorithm for computing the first local maximum is not publicly available.

In this study, an algorithm for computing the first local maximum in sequential data mining will be implemented. Chapter 2 defines some basic concepts. Chapter 3 reviews related work in the area of event data mining. Chapter 4 gives an efficient algorithm for computing the first local maximum and applies the algorithm on real ans synthetic datasets.

CHAPTER II

BASIC DEFINITIONS

Definition 1. Event type and event. The set of an e*vent types* (*E*) consists of all the possible types of the events that occurs in a particular background. An *event* is an occurrence of an *event type* (*e*) at a time (*t*) (denoted as (*e, t*)), in which $e \in E$, and *t* is the time when the event occurs [12; 13]. Different instances of the same *event type* at different times are considered as different *events*.

Definition 2. Event sequence and window. A serious of events ((e_1, t_1), (e_2, t_2), ..., (e_n, t_n)) forms an *event sequence* (*S*) in this research. For an *event sequence*, there is a starting time and an ending time, which are denoted as T_s and T_e respectively. [13]. All the *events* in the *event sequence* must happen in the time interval of $[T_s, T_e]$.

Figure 1 is an example of an event sequence. In this figure, *A, B, C, D, E, F* are the *event types*. (*A*, *35*), (*A*, *47*), (*A*, *59*) and (*A*, *65*) are different *events* in the sequence, but they are of the same *event type*, i.e. type *A*.

A *window w* is the interval between two events in an event sequence. In other words, a *window* is a part of a complete *event sequence*. *Windows* can overlap each other in the *event sequence*. The time difference between the starting and ending events (t_i and t_j) of a *window* is called the *width* of the *window*. A *Sliding window* is one of the methods used in sequential data analysis. [12].

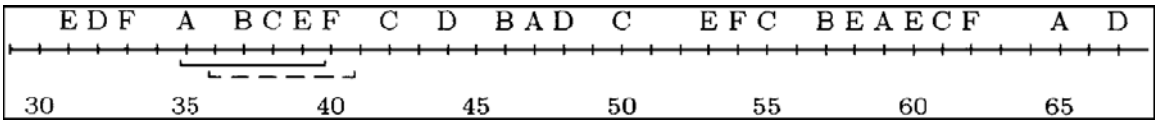

Figure 1. An example of an *event sequence* and two *windows* of width of 5 [12]. The two *windows* overlap each other.

Definition 3. Episode and episode size. An *episode* is an occurrence of one or more *events* in relative order in an event sequence. The order of the *events* is important, but these events do not necessarily occur consecutively. Other *events* can occur in between any two *events* of the *episode*.

The *size* of an episode is the number of events in the episode. Analysis on the frequently recurring patterns and their relationships in large sequential datasets is important in many applications.

For the *event sequence* example in Figure 1, (*E*), (*E*, *D*), and (*E*, *F*, *B*) are the *episodes*. The *sizes* of the *episodes* are 1, 2 and 3 respectively.

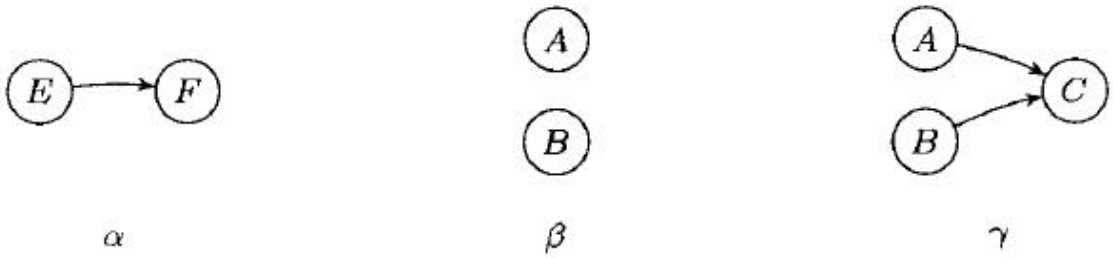

Figure 2: *Episode* α, *β* and γ. From [12]

Figure 2 shows different types of *episodes* according to the relationships among the occurrences of the *events* in the *episodes*. In a *serial episode*, the occurrences of the *events* in the *episode* happen one after another in order. *Episode* α is an example of *serial episodes*. *Event F* happens after *event E*. *Episode β* is a *parallel episode*. In a *parallel episode*, the relative order of *events A* and *B* is not important. They can happen

simultaneously, or one after another. The change of the order in which the *events* occur does not make a difference. *Episode* γ is an example of a *non-serial* and *non-parallel episode*. One *event* can only happen after the occurrence of two or more other *events* that form a *parallel episode* as the instances of *events* A and *B* in figure 1[12]. Actually we can consider the third type of episode as a combination of both *parallel episode* and *serial episode*.

The focus of this thesis is on *serial episodes*. In the following sections, the episodes all refer to serial episodes without special clarification.

Definition 4. Frequency of an episode. The *frequency* of an episode is the fraction of *windows* in which the episode occurs. For a given *event sequence* (*S*) and a *window* width (*win*), the *frequency* of an *episode* (α) in the event sequence is defined as the ratio of the number of *windows* in which the particular *episode* occurs to total number of the *windows* with width *win* in the *event sequence*. That is:

$$fr.(\alpha, S, win) = \frac{|\{w\,(\alpha \in W(S, win) \mid \alpha \text{ occurs in } w\}|}{|W(S,\ win)|} \quad [13]$$

Definition 5. Occurrence of an episode. An *occurrence* of an episode $\alpha=(e_1, e_2, \ldots, e_K)$ in an event sequence S is an ordered sequence of event $s' = \langle (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k) \rangle$, in which $s' \subseteq S$ and for every integer i between *1* and *k-1*, $0 < t_{i+1} - t_i \leq gapmax$. *Gapmax* is a user-defined integer threshold for the maximum time gap that can be allowed between any two consecutive *events* in occurrences of *episodes* in the sequential database.

The set of all occurrences of *episode* α in an event sequence *S* is denoted by *occ*(α, *S*). [t_1, t_k] stands for one instance of the occurrences of the *episode* α in *event sequence S* if s′ ($(e_1, t_1), (e_2, t_2),..., (e_k, t_k)$) is one member of *occ*(α, S) [13].

Definition 6. Width of an occurrence. Let $o = [t_s, t_e]$ be an *occurrence* of an *episode* in an *event sequence*. The time span of the occurrence, or $t_e - t_s$, is the width of the occurrence. It is denoted as *width*(*o*) [13].

From the definition, the widths of the occurrences of a single *event typ*e episode are always 0 since t_s is always equal to t_e in this case.

Definition 7. Minimal occurrence. For $[t_s, t_e]$, an occurrence of an *episode* α in the *event sequence S*, if no other occurrence $[t'_s, t'_e]$ of the episode α can be found such that $[t'_s; t'_e] \subset [t_s; t_e]$, then the occurrence of $[t_s, t_e]$ is a *minimal occurrence* of *episode* α in the *event sequence S*. The set of all *minimal occurrences* of episode α in the *event sequence S* is denoted by *mo*(α, S) [13]. In other words, if $[t_s, t_e]$ is a minimal occurrence of an *episode* α in *S*, then episode α should not occur in any proper sub-window of $[t_s, t_e]$.

The set of all *minimal occurrences* of an *episode* α in an e*vent sequence S* having a width of *w* is denoted by *mo*(*α, S, w*). *Occ*(*α, S, w*) is the set of all the *occurrences* of an *episode* α in an *event sequence S* with width of *w*.

From the definition of the minimal occurrence, it can be deduced that the *minimal occurrence* of an *episode* α in an *event sequence S* has the least width among all the occurrences that share the same starting point with the *minimal occurrence*.

Definition 8. Support of an episode. The *support* of an *episode* α in an *event sequence S* for a width *w* (*support* (α*, S, w*)) is the total number of *minimal occurrences*

of the *episode* α in *event sequence S*. The widths of the occurrences are not greater than width *w*. [13].

Lemma 1. For *episode* α and *event sequence S*, if $w_2 > w_1$, then *support* (α, *S*, w_1) $\leq$ support (α, *S*, w_2). *Support* (α, *S*, *w*) monotonously increases with the increase of *w* until to a maximum value, then the value of *support* (α, *S*, *w*) will remain the same even if *w*'s value increases.

Lemma 2. *support* (α, *S*, *w*) will be at its maximum value when w is equal to or greater than α * *gapmax*.

Definition 9. Prefix and suffix of an episode. The *suffix* of *episode*α is the *episode* that has the last *event* in the *episode*α. The width of the *suffix* of an episode is always equal to 1. For example, the *suffix* of *episode* (*A*, *B*, *C*) is *episode* (*C*). The *suffix* of episodeα is donated as *suffix*(α).

The *prefix* of *episode*α (*prefix*(α)) is the *episode* that has the all the *events* in episode α except the last one. The order of the *events* in the *prefix* is the same as *episode*α. The *prefix* of *episode (A*, *B*, *C*) is episode (*A*, *B*).

Definition 10. Episode rule. For *episodes* α and β, if *episode* α is the *prefix* of *episode* β, or *prefix* (β) = α, an *episode rule* built on *episodes* α and β is $\alpha \Rightarrow$ *suffix*(β) since α = *prefix* (β), $\alpha \Rightarrow$ *suffix*(β) is *prefix*(β) $\Rightarrow$ *suffix*(β) [13]. This actually forms the episode β.

Definition 11. Confidence of an episode rule. *Confidence* is the ratio of the *supports* of the two *episodes* in an *event sequence*. It could be understood as the probability of one *episode* happenning when another *episode* occurs. For an *episode rule* $\alpha \Rightarrow \beta$, the

support of an *episode rule* is *Support*($\alpha \Rightarrow$ *suffix*(β), S, w) = *Support(β, S, w)* since α = *prefix*(β) [13], and the *confidence* of an *episode rule* is defined as follows:

$$Confidence\ (\alpha \Rightarrow suffix(\beta), S, w) = \frac{\text{support}(\beta, S, w)}{\text{support}(\alpha, S, w)}$$

For *an episode rule*, its *confidence* in an *event sequence* changes with the value of the window width.

Definition 12. Minimal prefix occurrence of an episode. For an *occurrence O* of *episode* α in *event sequence* S, if we cannot find an instance of *mo*(*prefix*(α, S)) that occur inside of *O*, then we call it the *minimal prefix occurrence* of *episode* α, or an element in *mpo*(α, *S*). *mpo*(α, *S*) stands for the set of all *minimal prefix occurrences* of *episode* α in *event sequence* S.

Definition 13. Cluster of the minimal prefix occurrence of an episode. All the *minimal prefix occurrences* of *episode* α in event sequence *S* can be grouped by their starting position in the sequence. A *cluster* of the *minimal prefix occurrences* of an *episode* is the one or more *minimal prefix* occurrences of the episode that share the same starting position in the *event sequence*. *cmpo*(α, *S*, t_s) is used to denote the set of all the minimal prefix occurrences of episode α in the event sequence *S* in the cluster that starts at t_s.

Lemma 3. The shortest occurrence of *cmpo*(α, *S*, t_s) must be an occurrence of *mo* (α, *S*).

Lemma 4. if [t_s, t_e] is an occurrence of *mo*(α, *S*), then [t_s, t_e] is an occurrence of *cmpo* (α, *S*, t_s).

From lemma 3 and lemma 4, we can conclude that all occurrences of *mo*(α, *S*) are

included in *mpo*(α, *S*), and for every *cmpo*(α, *S*, t_s), there is an occurrence that is an occurrence of mo(α, *S*) and only the shortest occurrence of *cmpo*(α, *S*, t_s) is the occurrence of *mo*(α, *S*).

Definition 14. Local maximum (*LM*) and first local maximum (*FLM*). These two concepts are based on the changes of confidence of an *episode rule* with window width.

An episode rule *r* is said to have a *LM* (*Local Maximum*) for a given width *i* on event sequence *S* if the three following properties are satisfied [13]:

- $Confidence(r, S, i) \geq \gamma \wedge Support(r, S, i) \geq \delta$, where γ and δ are the thresholds for the confidence and support respectively. That means episode rule must be frequent and *prefix*(*r*) => *suffix*(*r*) are highly related.
- $\forall j,\ j < i\ \wedge\ Support(r, S, j) \geq\ \delta) \Rightarrow Confidence\ (r, S, i) > Confidence(r, S, j)$. The *Confidence* (*r*, *S*, *i*) at *LM* is not less than the confidence of the width in a range that is less than the width at *LM*.
- $\exists j;\ i < j \wedge Confidence(r, S, j) \leq Confidence(r, S, i)$- (*decoRate* * *Confidence*(*r*, *S*, *i*)) with *decRate* a decrease threshold defined by the user, and $\forall\ k;\ i < k < j\ )\ Confidence(r, S, k) \leq Confidence(r, S, i)$. In other words, the *Confidence*(*r*, *S*, *i*) at *LM* is significantly greater than the confidence of the width in a range that is greater than the width at *LM*.

Not all episode rules in any event sequence have *LM*s. If and only if there is at least one local maximum for the episode rule in the event sequence, the first one is called the first local maximum (*FLM*). An episode rule that has FLM width is called an FLM rule.

CHAPTER III

RELATED WORK

N. Meger and C. Rigotti [13] introduced the concepts of *Local Maximum* (*LM*) and the *First Local Maximum* (*FLM*) for constraint-based sequential pattern mining and provided the *WinMiner* algorithm for extracting the frequent episodes and the episode rules in a sequential event dataset. However, the details for searching FLM rules in a sequence database and computing the FLM window width for the episode rules have not been publicly available. Constraints include but are not limited to minimum support, minimum confidence, and gapmax. Minimum support can be used to judge if an episode is frequent in a sequence dataset. Minimum confidence is a value that tells if two episode rules are highly related. Gapmax specifies the maximum time difference between two consecutive events of an episode in a sequential dataset. The intuitive idea for this constraint is that the relationship between two events fades or weakens if the time elapsed between them increases. Two consecutive events are considered as not highly related and cannot form a valid episode if the time gap between them is greater than the gapmax value. The values of these constraints in data mining are usually fed in by the users according to their experience or the aim of their mining. The above constraints can improve the quality of the mining results and enhance the mining efficiency.

FLM and some related concepts are important in sequential pattern mining. The FLM window width of an episode rule can provide a reference for selecting an optimal window

size in the data mining. According to N. Meger and C. Rigotti's research, FLM rules don't exist in random datasets and can only be found in real-life ones.

Algorithm 1 (WinMiner)

Input: *S* is an event sequence and *E* is the set of event types. Other than these two data, δ, γ, decRate and *gapmax* are also required by the algorithms. δ is the minimum support threshold. γ is minimum confidence threshold. decRate is the minimum threshold for significant decrease of the confidence and *gapmax* is the maximum gap that is allowed between two consecutive events in an episode.

1. let $L_1 := \varnothing$;
2. for all $e \in E$ do
3. let *x.Pattern* := *e*
4. Let *x:Occ* := *scan(S, e)*
5. if | *x:Occ*| >= δ
6. let $L_1 := L_1 \cup \{x\}$
7. fi
8. od
9. for all $x \in L_1$ do
10. *exploreLevelN*(x, L_1)
11. od

x:Pattern is the episode rule and *x:Occ* is the support of the corresponding episode rule and they are called an *E*/*P* pair. This algorithm is based on *WinEPI*. It scans the *event sequence S* and counts the support for the single event episodes. When the support of a single event episode is not less than the minimum support value δ, the single event

episode is added into L_1. L_1 will have all the *MPO* of frequent episodes of size 1. For each member in L_1, call the *exploreLevelN* function.

Algorithm 2 (*exploreLevelN*)

Input*: x* an *E/O-pair, and L_1 the set of E/O-pairs of frequent episodes of size 1*. The value of δ, γ, decRate and *gapmax* in algorithm 1 need to be passed to this algorithm.

1. for all $y \in L_1$ do
2. let z := *join*(x, y)
3. if |*z:Occ*| >= δ
4. *findFLM*((*x:Pattern*) → *suffix*(*z:Pattern*), *x:Occ*, *z:Occ*)
5. exploreLevelN(z, L_1)
7. fi
8. od

The algorithm for the *exploreLevelN* function uses a single event episode in L_1 as the seed to generate the *MPOs* for the wider episodes by calling the *join* function to append an event type in L_1. Each time, the width of the episode increases by 1. When the *MPO* of the wider episode is available, the *findFLM* function is called to calculate the *FLM* if it exists.

Algorithm 3 (*join*)

Input: *x and y, two E/O-pairs, each contains an episode and its set of mpo, and y corresponds to an episode of size 1.* The value of δ, γ, decRate and *gapmax* in algorithm 1 and 2 need to be passed to this algorithm.

Output: *z, an E/O-pair containing the episode x:Pattern → y:Pattern and its set of mpo.*

1. let $z{:}Pattern := x{:}Pattern \rightarrow y{:}Pattern$

2. let $z{:}Occ := \varnothing$;

3. for all $(t_s; T) \in x{:}Occ$ do

4. let $L := \varnothing$;

5. for all $t \in T$ do

6. let $EndingTimes := \{t'_s \mid \exists\, [t'_s, T'] \in y{:}Occ$ such that

$t'_s > t_s \wedge t'_s - t \leq gapmax \wedge (t_1, T'') \in x{:}Occ; t_s < t_1 \Rightarrow \forall T_2 \in T''; t'_s \leq t_2$

7. let $L := L \cup EndingTimes$

8. od

9. if $L \neq \varnothing$;

10. let $z{:}Occ := z{:}Occ \cup \{(t_s; L)\}$

11. fi

12. od

This algorithm combines the two episodes *x:Pattern* and *y:Pattern* into a wider episode *z:Pattern*. The MPO of *z:Pattern, i. e. z.Occ*, is computed according to the MPO instances of episodes *x:Pattern* and *y:Pattern* and the definition of MPO.

CHAPTER IV

A STUDY OF THE FLM OF CONFIDENCE IN EPISODE RULE ANALYSIS

4.1 The Algorithm and Implementation for Computing An Episode's FLM

As previously mentioned, N. Meger and C. Rigotti [13] introduced the FLM concept but the algorithm for calculating the FLM was not publicly published. According to the FLM definition, an algorithm for computing the FLMs is proposed and implemented.

Algorithm 4 (*FindFLM*)

Input*: x* and *z* are two *E/O-pairs,* and *x:pPattern = prefix (z:Pattern)*.

1. for *j* = 0 to *eventNumber* do
2. if *support*((*z:Pattern, S, j*) >= δ
3. break
4. fi
5. od
6. if *j = eventNumber*
7. There is no FLM
8. fi
9. let *currentConfidence* = 0.0
10. let *nextConfidence* = 0.0

11. let *previousConfidence* = 0.0

12. let *maxWidth* = |*z:Pattern*| * *gapmax*

13. for *i* to *maxWidth* do

14. *nextConfidence* = *confidence*(*x:pattern* ⇒ *suffix*(*z:pattern*), *S, j*)

15. if (*nextConfidence* > ϒ and *nextConfidence* > *previousConfidence*)

16. *currentConfidence* = *previousConfidence* = *nextConfidence*

17. for *j* = *j*+1 to *maxWidth* do

18. *nextConfidence* = *confidence*(*x:pattern* ⇒ *suffix*(*z:pattern*), *S, j*)

19. if *nextConfidence* > *previousConfidence*

20. *currentConfidence* = *nextConfidence*

21. break

22. else

23. if it is a significant drop of confidence

24. FLM = *j*

25. fi

26. *previousConfidence* = *nextConfidence*

27. od

28. fi

29. if found FLM

30. break;

31. fi

32 . od

As mentioned before, the local maximum (LM) for episode rule *r* in sequence *S* at

width i must satisfies the following four requirements:

1. *Support*(r, S, i) $\geq \delta$, or episode rule r in sequence S must be frequent at window width i.

2. *Confidence*(r, S, i) $\geq \gamma$, or *prefix*(r)=> *suffix*(r) must be highly related in sequence S

3. The *confidence*(r, S, i) at window width i is not less than that at the width that is less than i.

4. There is a window width j and $j > i$, *confidence*(r, S, j) is significantly less than *confidence*(r, S, i). For all m, $i < m < j$, *confidence*(r, S, m) < *confidence*(r, S, i).

The above algorithm for *findFLM* was proposed according to the above LM window width properties. The *eventNumber* is the length of sequence S. Line 1 to 8 check the *support*((*z:pattern, S, j*) and make sure that the episode rule is frequent that is the first requirement for LM width . Episode rule r does not have LMs if the value of the support of rule does not pass the user defined support threshold δ when j reaches its maximum value. From the definition of support, if *support*((*z: pattern, S, j*) is greater than threshold δ, *support*((*z:pattern, S, m*) must be not less than δ for any m if $m >= j$.

In line 12 and 13, *maxWidth* is the possible maximum value of the LM window width for the episode *z: pattern*. According to lemma 2, both *support* ((*prefix*(*z:pattern*)*, S, j*) and *support*((*z: pattern, S, j*) stay at their maximum value respectively when $j \geq$ |*z:Pattern*| * *gapmax*. The value of *confidence*(*x: pattern* $\Rightarrow$ *suffix*(*z:pattern*), *S, j*) does not change when $i \geq$ *maxWidth*.

Line 15 makes sure the *Confidence*(*x: pattern* $\Rightarrow$ *suffix*(*z:pattern*), *S , j*) for possible LM window width j is beyond the threshold of γ. This is the second requirement for the

LM window width list above.

Lines 19 to 21 check if the third requirement for the FLM window width is met. Lines 23 to 24 search for a significant drop of confidence. The FLM is found if a significant drop is found.

In the implementation, the *MPO* for episode *r* in sequence *S* was organized as the list of *MPO* clusters. According to lemmas 3 and 4, only the shortest occurrence in a *MPO* cluster is the *MO* of the episode, and all the *MO* occurrences are also *MPO* occurrences. *support*((*z:Pattern, S, j*) can be easily computed by checking the shortest *MPO* in each cluster against the width *j*.

4.2 Datasets Used in This Research

Three datasets of different origins were used in my research.

The first dataset is the log of web access requests to the music machine web at http://machines.hyperreal.org. The log dataset for the web access requests February 12, 1997 can be found at http://www.cs.washington.edu/ai/adaptive-data/m.970212.gz. The first 2000 entries in the file were used in the research. The second dataset was randomly generated. The third dataset was also randomly generated but some correlations between the events were added.

According to introduction of the log, every event entry of the web access request dataset has four parts that were in the format of

O:<origin> || T:<time> || U:<url> || R:<referrer>

O is the origin, or the ID of the machine that sent the web access request. T stands for the time when the request was received. U is the URL of the requested webpage. The

URLs in the dataset are considered as event types in the sequential pattern research. R is the reference page for the request. The reference record may be blank if it is not applicable. The reference information was ignored in my study. After careful research on the dataset, the data entries were grouped by the machine ID, but the entries for each machine ID are in the order of time. An example of the data entries is:

O:00000000000000002335 || T:1997/02/14-20:02:36 ||
U:/machines/categories/software/ || R:http://www.synthzone.com/software.htm

The information of events that were used in this research are the times and event types or the URLs. The event data format that my WinMiner implementation can recognize is in the format of:

<ET> <T>

where ET is the event type of the event and T is the time the event occurred. A program was coded to transform the web access request dataset to the format of <ET> <T>. In the transformation, each webpage URL that requested in the dataset was assigned a unique short symbol in the range from "AAAA" to "ZZZZ" so that it could be more straightforward in reading. The time of the first access request entry in the dataset is set to the base time 0. If the machine ID of an access request is the same as the machine ID of the previous access request, the time of the access request is equal to the sum of the time of the previous request and the time gap of the two requests. Otherwise, it equals the sum of the previous access time and a user-selected integer (100000 is used in this research) no matter what the elapsed time is between the two requests. The integer selected by the user must be greater than the user-specified *gapmax* value so that access request events from different machines can be differentiated in the analysis. According to

the definition of episode occurrence, any two consecutive events that belong to two different machine IDs will not form a valid episode occurrence since the elapsed time gap between them is larger than *gapmax*. A total of 41 event types (URLs) can be found in the 2000 event sequence dataset and the largest time gap between two consecutive requests from the same machine in the dataset is 36740.

The following are the first four data entries in my research. This shows how the data were converted:

O:00000000000000000000 || T:1997/02/12-16:59:06 || U:/music/machines/ || R:http://www.csee.usf.edu/~gould/synth.html

O:00000000000000000000 || T:1997/02/12-16:59:40 || U:/music/machines/samples.html || R:http://hyperreal.com/music/machines/

O:00000000000000000000 || T:1997/02/12-17:00:16 || U:/music/machines/manufacturers/Roland/TR-909/samples/ || R:http://hyperreal.com/music/machines/samples.html

O:00000000000000000001 || T:1997/02/12-17:01:38 || U:/music/machines/manufacturers/Sequential/Tom/samples/ || R:http://hyperreal.com/music/machines/samples.html

O:00000000000000000001 || T:1997/02/12-17:02:37 || U:/music/machines/samples.html || R:-

After my transformation, the above entries became:

AAAA 0

AAAB 34

AAAC 70

AAAD 100070

AAAB 100129

The time for the first access request is 0 and it became the base time for calculating the time for other requests. The first three requests were sent from the same machine and their time gaps were 34 and 36 respectively. The time for the second request was 34 and that for the third request was 70. The fourth and fifth requests were from the same machine, which was different from the machine that sent the first three requests, so the time gap between the third and the fourth requests was 100000.The second and the fifth requested pages are the same one. The symbols for the URLs of these two requested pages are the same after conversion.

The second dataset was randomly generated without any correlation introduced. The pseudo code for the generation of the random dataset is:

Input: *nEventType* is the number of possible event types in the generated dataset, *nTimeGap* is the maximum time gap between any two consecutive events and *nEventNumber* is the event number of the dataset.

1. for i = 0 to *nEventNumber* -1 do
2. e*ventType* = generateRandType(*nEventType*)
3. t*imeGap* = generateRandGap(*nTimeGap*)
4. printout(*eventType*, *timeGap* + the time of the previous event)
5. od

Where generateRandType(*nEventType*) and generateRandGap(*nTimeGap*) generate two integers whose maximum value are *nEventType* and *nTimeGap* respectively.

The third dataset used in my research is randomly generated with some correlation added. Its difference with the second dataset is that correlations of event types were added. For example, event type of an event was always B if the event type of the previous event was C and such. The stronger are this kind of correlations, the stronger the relationships among events will be.

For better comparison with the web access request dataset, the total number of event types in the two randomly generated datasets was selected as 41 and the number of events in each dataset is 2000.

4.3 Conclusion

N. Meger and C. Rigotti [13] claimed that FLM rules cannot be found in randomly generated datasets and a local maximum of confidence could only be found in real-life datasets. Lack of FLM rules in a sequential dataset means the lack of the dependencies among the events in a sequential dataset.

Table 1 sumarizes my experimental results on the three datasets. It shows the number of FLM rules for the three datasets on different support, confidence threshold and the value of gapmax. However, the confidence threshold γ and decRate value were fixed at 0.1 and 0.3 respectively. From the table we can draw some conclusions:

1. For the same dataset, the number of FLM rules increases with the increase of gapmax value. Take the web access request dataset as the example, when δ is set to 2, the number of FLM rules in the random generated dataset with correction increases from 8 when the gapmax is 10 to 388 when gapmax is 50.

2. For the same dataset, the number of FLM rules decreases with the increase of

support threshold value δ. The number of FLM rules increased from 29 to 141 when the δ value decreased from 3 to 2 with gapmax is set at 500.

Dataset	δ *	gapmax	FLM-rule Number
Web Access Request Dataset	2	10	8
Web Access Request Dataset	2	50	388
Web Access Request Dataset	3	50	31
Web Access Request Dataset	3	500	533
Random Dataset With correlation	2	50	0
Random Dataset With correlation	2	10	0
Random Dataset With correlation	3	50	0
Random Dataset With correlation	2	500	141
Random Dataset With correlation	3	500	29
Random Dataset Without correlation	2	10	0
Random Dataset Without correlation	2	50	0
Random Dataset Without correlation	2	500	1
Random Dataset Without correlation	3	50	0
Random Dataset Without correlation	3	500	0

Table 1. The Number of FLM Rules for Different Datasets
δ: The threshold for minimum support;

In data mining, loosening constraints, for example increasing gapmax values or decreasing the support threshold causes more episode rules to be counted as valid frequent and highly related ones. This in turn increases the number of FLM rules.

3. In many cases FLM rules cannot be found in randomly generated datasets without correlations. Only one FLM-rule exists in the dataset without correlations in this research, and even the data mining constraints were very loose. But correlations added in the data generation process increase the number of the FLM rules dramatically even it had far less

FLM rules than the datasets in real life.

The experiments in my research show that no or rare FLM rules in a sequential dataset indicate that it is random dataset. The events in the dataset have no relationships or dependencies. However, the number of FLM rules can be a good parameter to judge how strong the dependencies among the events in datasets are.

REFERENCES

[1] R. Agrawal, R. Srikant, "Mining Sequential Patterns**",** *Proceedings of the 11th IEEE International Conference on Data Engineering*, 1995, pp. 3 –14.

[2] T. Abraham, "Event Sequence Mining to Develop Profiles for Computer Forensic Investigation Purposes," *Proceedings of the Fourth Australasian Information Security Workshop (Network Security)*, vol.54, 2006, pp. 145 -153

[3] J. Baixeries, J. Casas-Garriga, and J. Balcazar,. "Mining Unbounded Episodes from Sequential Data," *NeuroCOLT Technical Reports 2001*, 2001, available from http://www.neurocolt.com/tech_reps/2001/seqs.ps.

[4] R. Brause, T. Langsdorf, and M. Hepp, "Neural Data Mining for Credit Card Fraud Detection", *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, 1999, p. 103.

[5] X. R. Jiang, and L. Gruenwald, "Microarray Gene Expression Data Association Rules Mining Based On JG-Tree," *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03*), 2003, pp. 27-31.

[6] J. Li, L. Wong, and Q. Yang, "Data Mining in Bioinformatics," *IEEE Intelligent Systems, IEEE Computer Society*, 2005, pp.16-18.

[7] S. V. Nath, "Crime Pattern Detection Using Data Mining**",** *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2006, pp. 41- 44

[8] T. M. Mitchell, "Machine Learning and Data Mining," *Communications of the ACM*, vol. 42, no. 11, 1999, pp. 30-36.

[9] H. Mori, N. Kosemura, T. Kondo, and K. Numa, "Data Mining for Short-term Load Forecasting," *Power Engineering Society Winter Meeting,* 2002, pp. 623-624.

[10] H. Mannila, H. Toivonen, and I. Verkamo, "Discovering Frequent Episodes in Sequences," *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD '95)*, 1995, pp. 210–215.

[11] H. Mannila, and H. Toivonen, "Discovering Generalized Episodes Using Minimal Occurrences," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996, pp. 146-151.

[12] H. Mannila, H. Toivonen, and A. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery,* vol. 1, no. 3, 1997, pp. 259-298.

[13] N. Meger, and C. Rigotti, "Constraint-Based Mining of Episode Rules and Optimal Window Sizes," Proceedings *of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'04)*, 2004, pp. 313-324.

[14] N. Meger, C. Leschi, N. Lucas, and C. Rigotti, "Mining episode rules in STULONG dataset," Proceedings of the ECML/PKDD2004 Discovery Challenge, 2004, pp. 1-12.

[15] W. Sia, and M. M. Lazarescu, "Clustering Large Dynamic Datasets Using Exemplar Points," *Proceedings of 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*, 2005, pp.164-173.
[16] J. K. Ting, M. K. Ng, H. Rong, and J. Z. Huang, "Statistical Models for Time Sequences Data Mining," *Proceedings of IEEE International Conference on Computational Intelligence for Financial Engineering*, 2003, pp. 347- 354.
[17] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations* (2000), Morgan Kaufmann.
[18] W. Wong, A. Moore, G. Cooper, and M. Wagner, "Rule-Based Anomaly Pattern Detection for Detecting Disease Outbreaks," *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002, pp. 217-223.
[19] Y. Y. Yao, "On Modeling Data Mining with Granular Computing," *Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC 2001)*, 2001, pp. 638–643.
[20] M. J. Zaki, C. T. Ho, and R. Agrawal, "Parallel Classification for Data Mining on Shared-Memory Multiprocessors", *IBM Technical Report*, 1998, available from. http://citeseer.ist.psu.edu/zaki98parallel.html
[21] Q. Zheng, K. Xu, W. Lv, and S. Ma, "Intelligent Search of Correlated Alarms from Database Containing Noise Data," *Proceedings of 8th IEEE/IFIP Network and Operations Management Symposium* (NOMS), 2002, pp. 405-419.
[22] Q. Zheng, K. Xu, W. Lv, and S. Ma, "Intelligent Search of Correlated Alarms for GSM Networks with Model-based Constraints," *Proceedings of 9th IEEE International Conference on Telecommunications (ICT)*, vol. 2, 2002, pp. 635-645.

APPENDIX

PROGRAM SOURCE CODES

episode.h

```
#ifndef _EPISODE_H
#define _EPISODE_H

/*****************************************************************************
                Macro define section
*****************************************************************************/
#define MAX_PATH_LENGTH 256
#define MAX_LINE_LENGTH 256
#define MAX_EPISODE_LENGHTH 40

#define DEFAULT_GAMA 0.5
#define DEFAULT_DELTA 5
#define DEFAULT_MAXGAP 10
#define DEFAULT_DECRATE 0.2
#define RESULT_FILE "result"
#define DEFAULT_EVENT_SIZE 1000
#define DEFAULT_EVENT_TYPE_NUMBER 100

#ifdef WIN32
  #define getpid _getpid
#endif

/*****************************************************************************
                include section
*****************************************************************************/
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>

#ifdef WIN32
  #include <process.h>
#else
  #include <unistd.h>
#endif

#include "linkedList.h"

/*****************************************************************************
                Structure definition section
*****************************************************************************/
typedef struct _EVENTTYPE
{
  char * pszEvent;              /*Event type names*/
  char * pszDescription;        /*descriptions for the event type*/
}EVENT_TYPE, *PEVENT_TYPE;

typedef struct _EVENT
{
  int   nPosInEventTypeList;    /*use the position in Event type list*/
```

```
  long  nPosInEventList;          /*Position in the sequence (event list)*/
  long  nTime;                    /*The date when the event happens.*/
}EVENT, *PEVENT;

typedef struct _MPOCLUSTER
{
  int       nStartPosInEventList;          /*The start position in the sequence*/
  PLINKEDLIST pEndingPosInEventList;  /*The list of ending positions*/
}MPOCLUSTER, *PMPOCLUSTER;

/*This structure is for the mpo that share the same start point*/
typedef struct _MPO
{
  PLINKEDLIST pMPOClusterList;    /*The MPO cluster*/
  PLINKEDLIST pEpisodeEventList;  /*The list of event index in the episode*/
  int       nFLM;                 /*the FLM for this episode*/
  long      nTotalOccurrence;
}MPO, *PMPO;

typedef struct _SEQUENCE
{
  char *     pszDescription;          /*descriptions for the sequence*/
  PEVENT_TYPE pEventTypeArray;    /* The event types array, EVENT_TYPE*/
  int       nEventTypeArraySize;     /*The size of the array*/
  int       nEventTypeNumber;        /*The used space of the allocated space*/
  PEVENT     pEventArray;            /*The event array of the sequence */
  long      lEventArraySize;         /*The allocated size of the array*/
  long      lEventNumber;            /*The used space of the used event array*/
  PLINKEDLIST *ppMPOList;            /*The pointer arrary to mpo list */
  int       nMPOListArraysize;       /*the size of the array*/
  int       nMPOListNumber;          /*The used space of the MPO pointer array*/
  char      szInputFilePath[MAX_PATH_LENGTH];   /*The input file path*/
  char      szOutputFilePath[MAX_PATH_LENGTH];  /*the output file path*/
  int       nDelta;            /*The integer for delta threshhold*/
  int       nMaxGap;           /*the maxgap*/
  float     fGama;             /*The float for gama threshhold*/
  float     fDecRate;          /*The user defined float*/
}SEQUENCE, *PSEQUENCE;


/*****************************************************************************
              function prototype list
*****************************************************************************/
void printCommandFormat(void);
void initializeStructure(PSEQUENCE pStructSequence);
int parseCommandLine(PSEQUENCE pStructSequence, int argc, char *argv[]);
void ReadInputFile(PSEQUENCE pStructSequence);
int searchEventType(PSEQUENCE pStructSequence,char* pszEventType);
void addEventToArray(PSEQUENCE pStructSequence, long nIndex,
          long nTime, int nTypeIndex);
int addEventType(PSEQUENCE pStructSequence,char* pszEventType, char* pszDesc);
int addMPO(PSEQUENCE pStructSequence, PLINKEDLIST *ppMPOList,
        PLINKEDLIST pEventLinkedList, int nStartIndex,  int nEndIndex);
```

```
void checkDeltaForMPOList(PSEQUENCE pStructSequence, PLINKEDLIST pMPOList);
int checkDeltaForAMPO(PSEQUENCE pStructSequence, PLINKEDLIST pMPOList,
                PNODE ptrMPONode);
int checkMPOCondition(PSEQUENCE pStructSequence, PMPO pStructMPO,
        int nStartIndex, int nEndingIndex);
void removeMPO(PSEQUENCE pStructSequence, PNODE ptrMPONode);
void exploreLevelN(PSEQUENCE pStructSequence, PLINKEDLIST pMPOList);
PNODE join(PSEQUENCE pStructSequence, PMPO pStructMPOX, PMPO pStructMPOY,
         PLINKEDLIST *ppMPOList);
void findFLM(PSEQUENCE pStructSequence, PMPO pStructMPOX, PMPO pStructMPOZ);
int calculateSupport(PSEQUENCE pStructSequence, PMPO pStructMPO, int i);
float calculateConfidence(PSEQUENCE pStructSequence, PMPO pStructMPOZ,
        PMPO pStructMPOX, int i);
void printMPO(PSEQUENCE pStructSequence, PNODE ptrMPONode);
void printFLM(PSEQUENCE pStructSequence);
void printEpisode(PSEQUENCE pStructSequence, PNODE ptrMPONode);
void freeSpaceAndExit(PSEQUENCE pStructSequence, RETURN_TYPE nReturnType);
void printLogFileHeader(PSEQUENCE pStructSequence);
void printOutputFileHeader(PSEQUENCE pStructSequence);
PNODE formAnMPOClusterNode(PSEQUENCE pStructSequence, long nStartIndex);

#endif
```

episode.cpp

```
/*****************************************************************************
                Macro define section
*****************************************************************************/

/*****************************************************************************
                include section
*****************************************************************************/
#ifdef WIN32
  #include <winsock2.h>
#else
  #include <unistd.h>
#endif

#include <time.h>

#include "episode.h"
#include "getopt.h"
#include "log.h"


/*****************************************************************************
                Global variables
*****************************************************************************/


/*****************************************************************************
                Local function prototype list
*****************************************************************************/

int main (int argc, char *argv[])
{
  SEQUENCE structSequence = {0};
  PMPO          pStructMPO    = NULL;
  PNODE         ptrMPONode    = NULL;
  int           i             = 0;

  initializeStructure(&structSequence);

  if (!parseCommandLine(&structSequence, argc, argv))
  {
    printCommandFormat();
    freeSpaceAndExit(&structSequence, RETURN_ERROR);
  }


  /*The following code do the job in Algorithm 1
  The E-O pairs are stored in structSequence.pMPOList*/
  ReadInputFile(&structSequence);

  /*Print the header information for output file*/
```

```
  printOutputFileHeader(&structSequence);

     checkDeltaForMPOList(&structSequence, *(structSequence.ppMPOList));

  ptrMPONode = (*(structSequence.ppMPOList))->pListNode;

 //  for(; ptrMPONode; ptrMPONode = ptrMPONode->pNextNode)
 //     printMPO(&structSequence, ptrMPONode);

  exploreLevelN(&structSequence, structSequence.ppMPOList[0]);

  printFLM(&structSequence);

  freeSpaceAndExit(&structSequence, RETURN_SUCCESS);

  return 0;
}

/*****************************************************************************
  Description:
    This function implementes the algoritm 2. The pEpisodeEventList in every
    of MPO node in pStructSequence->pMPOList equals to x. All the MPO in the
    linked list equal to the L1.

  Input:
    pStructSequence: The SEQUENCE structure that needs to be initialized

  Returned value:
       void
  History:
    04/09/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void exploreLevelN(PSEQUENCE pStructSequence, PLINKEDLIST pMPOList)
{
  int   nArraySize   = 0;
  int   i            = 2;
  PNODE ptrMPONodeX  = NULL;
  PMPO  pStructMPOX  = NULL;
  PNODE ptrMPONodeY  = NULL;
  PMPO  pStructMPOY  = NULL;
  int   bContinue    = 1;

  PNODE  pNewMPONode      = NULL;

  /*Start from 1 since we have the MPO for the episode of size of 1*/
  while(1)
  {

    /*if there are no more MPOs, don't need to expand it again*/
    if(bContinue == 0)
      break;

    pStructSequence->nMPOListNumber++;
```

```
if(pStructSequence->nMPOListNumber > pStructSequence->nMPOListArraysize)
{

  nArraySize = pStructSequence->nMPOListArraysize + MAX_EPISODE_LENGHTH;
  pStructSequence->ppMPOList
    = (PLINKEDLIST *)realloc(pStructSequence->ppMPOList,
      sizeof(PLINKEDLIST)*nArraySize);

  pStructSequence->nMPOListArraysize = nArraySize;
}

/*rest to 0*/
bContinue    = 0;

ptrMPONodeX = pStructSequence->ppMPOList[i-2]->pListNode;

for(; ptrMPONodeX; ptrMPONodeX = ptrMPONodeX->pNextNode)
{
  pStructMPOX = (PMPO)(ptrMPONodeX->pContent);

  ptrMPONodeY = pMPOList->pListNode;
  for (; ptrMPONodeY; ptrMPONodeY = ptrMPONodeY->pNextNode)
  {
    /*Reset it to NULL every time*/
    pNewMPONode = NULL;

    pStructMPOY = (PMPO)(ptrMPONodeY->pContent);

    /*The Number of the MO equals to the number of the MPOCluster.
      so, if the support is less than the minimum requirement, do
      not expanded it*/
    if(pStructMPOY->pMPOClusterList->nCount < pStructSequence->nDelta)
      continue;

    pNewMPONode = join(pStructSequence, pStructMPOX,
              pStructMPOY, &(pStructSequence->ppMPOList[i-1]));

    /*New MPO  was formed for the extended episode*/
    if (pNewMPONode)
    {
      /*if the new episode has no less occurrence than the Delta
      requirement, then calculate the FLM*/
      if (!checkDeltaForAMPO(pStructSequence,
            pStructSequence->ppMPOList[i-1], pNewMPONode))
      {
        bContinue    = 1;

        findFLM(pStructSequence, (PMPO)(ptrMPONodeX->pContent),
          (PMPO)(pNewMPONode->pContent));

      }
    }
```

```
        }
      }

     i++;

   }
}


/*****************************************************************************
   Description:
      This function finds the FLM. The episode in pStructMPOX should be the
      prefix of the episode in pStructMPOZ. Refer the article for the
      definition of "prefix"

   Input:
      pStructSequence:  The SEQUENCE structure that needs to be initialized
      pStructMPOX:      E-O pair x in the algorithm 2
      pStructMPOZ       E-O pair Z in the algorithm 2.

   Returned value:
      void

   History:
      04/10/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void findFLM(PSEQUENCE pStructSequence, PMPO pStructMPOX, PMPO pStructMPOZ)
{
   long  i = pStructMPOZ->pEpisodeEventList->nCount;
   int   nSupport = 0;
   float fConfidence = 0.0, fNextConfidence = 0.0, fPrevConfidence = 0.0;
   float fDifference = 0.0;
   int   bFoundFLM   = 0;
   long  lnMaxWinlength = 0;

   pStructMPOZ->nFLM = 0;

   /*1. Make sure the support is not less than the threshhold*/
   for(; i < pStructSequence->lEventNumber; i++)
   {
      nSupport = calculateSupport(pStructSequence, pStructMPOZ, i);

       if(!(nSupport < pStructSequence->nDelta))
          break;
   }

   /*if the end of the sequence is reached, that means there is no FLM*/
   if( i == pStructSequence->lEventNumber)
      return;

   /*2. Look for the largest confidence and then check if its width is
   the FLM*/
   lnMaxWinlength = pStructMPOZ->pEpisodeEventList->nCount * pStructSequence->nMaxGap;
```

```
  for(; i < lnMaxWinlength; i++)
  {
    fNextConfidence
      = calculateConfidence(pStructSequence, pStructMPOZ, pStructMPOX, i);

    /*Make sure the confidence is not less than the threshhold and
    keep the largest confidence*/
    if(!(fNextConfidence < pStructSequence->fGama) &&
      (fNextConfidence  > fPrevConfidence))
    {
      fConfidence = fPrevConfidence = fNextConfidence;

      for (i = i+1; i < lnMaxWinlength; i++)
      {

        fNextConfidence
          = calculateConfidence(pStructSequence, pStructMPOZ, pStructMPOX, i);


        if(fNextConfidence > fPrevConfidence)
        {
          /*Don't need to check the confidence threshhold since we know
           the confidence is larger than it*/
          fConfidence = fNextConfidence;
          break;
        }
        else
        {
          fDifference  = fConfidence - fNextConfidence;
          /*3. Make sure the it is a significant drop of confidence*/
          if(!(fDifference < pStructSequence->fDecRate * fConfidence))
          {
            pStructMPOZ->nFLM = i;
            break;
          }

        }

        fPrevConfidence = fNextConfidence;
      }

      /*Found FLM, exit the searching for FLM*/
      if(pStructMPOZ->nFLM != 0)
        break;
    }
  }
}

/*****************************************************************************
  Description:
    This function calculate the confidence of a episode in the sequence with
    width of i. Please refer the article for the definition of "confidence"
```

```
  Input:
    pStructSequence:  The SEQUENCE structure that needs to be initialized
    pStructMPO:       E-O pair x.
    i:                The width used in calculating support.

  Returned value:
    float

  History:
    04/12/2007: Created by Chuanwu (Steven) Xiong
*************************************************************************/
float calculateConfidence(PSEQUENCE pStructSequence, PMPO pStructMPOZ,
      PMPO pStructMPOX, int i)
{
  int nSupportZ = 0;
  int nSupportX = 0;

  nSupportZ = calculateSupport(pStructSequence, pStructMPOZ, i);
  nSupportX = calculateSupport(pStructSequence, pStructMPOX, i);

  return (float)nSupportZ/(float)nSupportX;
}

/*************************************************************************
  Description:
    This function calculate the support of a episode in the sequence with
    width of i. Please refer the article for the definition of "support"

  Input:
    pStructSequence:  The SEQUENCE structure that needs to be initialized
    pStructMPO:       E-O pair x.
    i:                The width used in calculating support.

  Returned value:
    int

  History:
    04/12/2007: Created by Chuanwu (Steven) Xiong
*************************************************************************/
int calculateSupport(PSEQUENCE pStructSequence, PMPO pStructMPO, int i)
{
  int       nSupport        = 0;
  PNODE     pMPOClusterNode   = NULL;
  PNODE     ptrNode         = NULL;
  PMPOCLUSTER pStructMPOCluster = NULL;
  int       nStartIndex = 0, nEngdingIndex = 0;
  int       nStartTime = 0,  nEndingTime = 0;

  pMPOClusterNode = pStructMPO->pMPOClusterList->pListNode;

  for (; pMPOClusterNode; pMPOClusterNode = pMPOClusterNode->pNextNode)
  {
```

```
    pStructMPOCluster = (PMPOCLUSTER)(pMPOClusterNode->pContent);

    /*Get the start time of the MPOCLUSTER*/
    nStartIndex = pStructMPOCluster->nStartPosInEventList;

    nStartTime = (pStructSequence->pEventArray + nStartIndex)->nTime;

    /*Get the information for the ending event. According to the definition
    of MO and MPO, we can assume that only the first in the MPO cluster can be
    the the mo. The support definition is based on MO not MPO.*/
    ptrNode =pStructMPOCluster->pEndingPosInEventList->pListNode;

    if(ptrNode->pContent)
    {
      nEngdingIndex = *((int*)(ptrNode->pContent));
      nEndingTime = (pStructSequence->pEventArray + nEngdingIndex)->nTime;
      if((nEndingTime - nStartTime) > i)
        continue;
      else
        nSupport++;

    }
    else
    {
      writeLog(ERROR_MSG, "The Invalid event node. File %s, line %d",
        __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }
  }

  return nSupport;
}



/*****************************************************************************
  Description:
    This function implementes the algoritm 3.

  Input:
    pStructSequence:  The SEQUENCE structure
    pStructMPOX:      E-O pair x in the algorithm
    pStructMPOY       E-O pair y in the algorithm. An episode of size 1

  Returned value:
      Return a pointer to the new MPO node joined from pStructMPO1 and
      pStructMPO2

  History:
    04/10/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
PNODE join(PSEQUENCE pStructSequence, PMPO pStructMPOX, PMPO pStructMPOY,
```

```
        PLINKEDLIST *ppMPOList)
{
  PNODE pTempNode        = NULL;

  PNODE ptrMPONodeX1     = NULL;
  PNODE ptrMPOClusterNodeX= NULL;
  PNODE ptrEndingPosNodeX = NULL;
  PMPOCLUSTER pStructMPOClusterX = NULL;

  PLINKEDLIST pEpisodeEventList  = NULL;
  PNODE ptrMPOClusterNodeY= NULL;
  PNODE ptrEndingPosNodeY = NULL;
  PMPOCLUSTER pStructMPOClusterY = NULL;
  PLINKEDLIST pEpisodeEventListY = NULL;

  int   bNewMPOFound  = 0;
  int   nStartIndexX  = 0,   nEndingIndexX  = 0;
  int   nStartIndexY  = 0,   nEndingIndexY  = 0;
  int   nStartTimeX   = 0,   nEndingTimeX   = 0;
  int   nStartTimeY   = 0,   nEndingTimeY   = 0;
  int   nGap  = 0;


  /*If the pMPOClusterList of pStructMPOx is NULL, then do nothing.*/
  if(!(pStructMPOX->pMPOClusterList))
    return NULL;

  /*Form the new episode*/
  pEpisodeEventList
    = duplicateLinkedList(pStructMPOX->pEpisodeEventList);
  addNodeToListTail(pEpisodeEventList,
    pStructMPOY->pEpisodeEventList->pListNode);

  ptrMPOClusterNodeX = pStructMPOX->pMPOClusterList->pListNode;

  for(; ptrMPOClusterNodeX;
        ptrMPOClusterNodeX = ptrMPOClusterNodeX->pNextNode)
  {
    pStructMPOClusterX =(PMPOCLUSTER)(ptrMPOClusterNodeX->pContent);

    /*The index of the start position  in the event list*/
    nStartIndexX = pStructMPOClusterX->nStartPosInEventList;

    /*Get the time of the start event of the episode*/
    nStartTimeX = (pStructSequence->pEventArray + nStartIndexX)->nTime;

    ptrEndingPosNodeX = pStructMPOClusterX->pEndingPosInEventList->pListNode;

    for (; ptrEndingPosNodeX; ptrEndingPosNodeX=ptrEndingPosNodeX->pNextNode)
    {

      /*The index of the ending position in the event list*/
      nEndingIndexX = *(int*)(ptrEndingPosNodeX->pContent);
```

```
        /*Get the time of the ending event of the episode*/
        nEndingTimeX = (pStructSequence->pEventArray + nEndingIndexX)->nTime;

        ptrMPOClusterNodeY = pStructMPOY->pMPOClusterList->pListNode;
        for(; ptrMPOClusterNodeY;
              ptrMPOClusterNodeY = ptrMPOClusterNodeY->pNextNode)
        {
          pStructMPOClusterY = (PMPOCLUSTER)(ptrMPOClusterNodeY->pContent);

          /*The index for the start position in the linked list*/
          nStartIndexY = pStructMPOClusterY->nStartPosInEventList;

          /*Get the time of the start event of the episode*/
          nStartTimeY = (pStructSequence->pEventArray + nStartIndexY)->nTime;

          /*Since pStructMPOY is a MPO of a single event. we can safely
          assume that every cluster has only one node in the
          endingPosLinkedList. The start and ending position index should
          also be of the same value*/
          nEndingIndexY = nStartIndexY;
          nEndingTimeY = nStartTimeY;

          if(nStartTimeY > nEndingTimeX &&
            (nStartTimeY - nEndingTimeX) < pStructSequence->nMaxGap)
          {
             if (checkMPOCondition(pStructSequence, pStructMPOX,
               nStartIndexX, nStartIndexY))
             {
               addMPO(pStructSequence, ppMPOList, pEpisodeEventList,
                  nStartIndexX, nStartIndexY);

               bNewMPOFound = 1;
             }
          }
        }
      }

    }

    /*Free the space*/
    destroyLinkedList(pEpisodeEventList);

    if(bNewMPOFound)
      return ((*ppMPOList)->pLastNode);
    else
      return NULL;
}

/*****************************************************************************
    Description:
      This function checks if the new occurrence satifies the MPO condition
      for all (t1, T1) in MPO set, if nStartIndex < t1, then for all t2 in T1,
```

```
    nEndingIndex <= t2

  Input:
    pStructSequence: The SEQUENCE structure
    pStructMPO:      The MPO structure for the paticular episode
    nStartIndex:     The start index of the new occurrence
    nEndingIndex:    The ending index of the new occurrence

  Returned value:
    1 if satified the requirement, 0 otherwise

  History:
    04/10/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
 int checkMPOCondition(PSEQUENCE pStructSequence, PMPO pStructMPO,
       int nStartIndex, int nEndingIndex)
 {
  PNODE pTempNode           = NULL;
  PNODE ptrMPONode          = NULL;
  PNODE ptrMPOClusterNode   = NULL;
  PNODE ptrEndingPosNode    = NULL;
  PMPOCLUSTER pStructMPOCluster = NULL;
  PLINKEDLIST pEpisodeEventList = NULL;
  int nReturn      = 1; /*Assume it is MPO first*/
  int nStartIndex1 = 0,   nEndingIndex1  = 0;

  /* Go through all the MPO cluster*/
  ptrMPOClusterNode = pStructMPO->pMPOClusterList->pListNode;
  for(; ptrMPOClusterNode; ptrMPOClusterNode = ptrMPOClusterNode->pNextNode)
  {
    pStructMPOCluster = (PMPOCLUSTER)ptrMPOClusterNode->pContent;
    nStartIndex1 = pStructMPOCluster->nStartPosInEventList;

    if(nStartIndex < nStartIndex1)
    {
      pEpisodeEventList = pStructMPOCluster->pEndingPosInEventList;
      ptrEndingPosNode = pEpisodeEventList->pListNode;

      for (; ptrEndingPosNode; ptrEndingPosNode= ptrEndingPosNode->pNextNode)
      {
        nEndingIndex1 = *((int*)(ptrEndingPosNode->pContent));

        /*The requirement for the MPO cannot be satisfied, then return
        the value*/
        if (nEndingIndex > nEndingIndex1)
          return 0;
      }
    }
  }

  return nReturn;
 }
```

```
/*****************************************************************************
  Description:
    This function initializes the sequence structure

  Input:
    pStructSequence: The SEQUENCE structure that needs to be initialized

  Returned value:
    int

  History:
    03/06/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/

void  initializeStructure(PSEQUENCE pStructSequence)
{
  int i = 0;

  memset(pStructSequence, 0, sizeof(SEQUENCE));


  pStructSequence->pEventArray
    = (PEVENT)malloc(DEFAULT_EVENT_SIZE*sizeof(EVENT));

  pStructSequence->lEventArraySize = DEFAULT_EVENT_SIZE;
  pStructSequence->lEventNumber = 0;

  pStructSequence->pEventTypeArray
    = (PEVENT_TYPE)malloc(DEFAULT_EVENT_TYPE_NUMBER*sizeof(EVENT_TYPE));
  pStructSequence->nEventTypeArraySize = DEFAULT_EVENT_TYPE_NUMBER;
  pStructSequence->nEventTypeNumber = 0;

  pStructSequence->ppMPOList
    = (PLINKEDLIST  *)malloc(sizeof(PLINKEDLIST)*MAX_EPISODE_LENGHTH);

  /*Initialize the array*/
  for (i = 0; i < MAX_EPISODE_LENGHTH; i++)
  {
    pStructSequence->ppMPOList[i] = initializeLinkedList(MPO_TYPE);

    if(pStructSequence->ppMPOList[i] == NULL)
    {
      writeLog(ERROR_MSG, "Memory error when i = %d. File %s, line %d. ",
        i, __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }
  }

  pStructSequence->nMPOListArraysize = MAX_EPISODE_LENGHTH;
  pStructSequence->nMPOListNumber = 0;

  pStructSequence->fGama = DEFAULT_GAMA;
```

```
  pStructSequence->nDelta = DEFAULT_DELTA;
  pStructSequence->nMaxGap = DEFAULT_MAXGAP;
  pStructSequence->fDecRate = (float)DEFAULT_DECRATE;

#ifdef WIN32
  sprintf(pStructSequence->szOutputFilePath, "log\\%s%d.log",
    RESULT_FILE, getpid());
#else
  sprintf(pStructSequence->szOutputFilePath, "log/%s%d.log",
    RESULT_FILE, getpid());
#endif

  printLogFileHeader(pStructSequence);
}


/*****************************************************************************
  Description:
    This function parses the command line and make sure the application
    gets all needed paramters and use the command input to populate
    the SEQUENCE structure

  Input:
    pStructSequence: The structure will be populated
    argc: The number of the arguments from the command
    argv: The arguments from the command line

  Returned value:
    0 if there are errors, 1 success

  History:
    03/06/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int parseCommandLine(PSEQUENCE pStructSequence, int argc, char *argv[])
{
  extern char *optarg;
  int nOption    = 0;
  int inttemp    = 0;
  FILE* ptrFile  = NULL;
  int bInputFile = 0;

  if (argc < 2)
  {
    writeLog(ERROR_MSG, "More arguments are expected. File %s, line %d",
      __FILE__, __LINE__);

    return 0;
  }

  /* read input parameters and specify input */
  while ((nOption = getopt(argc, argv, "I:i:G:g:D:d:O:o:R:r:M:m:"))!= EOF)
  {
    switch(nOption)
```

```
{

  /*The episode application must have a accessible input file*/
  case 'I':
  case 'i':
    sscanf(optarg, "%s", pStructSequence->szInputFilePath);

    if (strlen(pStructSequence->szInputFilePath) == 0)
    {
      writeLog(ERROR_MSG, "No input data file. File %s, line %d. ",
        __FILE__, __LINE__);

      return 0;
    }

    if(fopen(pStructSequence->szInputFilePath, "r") == NULL)
    {
      writeLog(ERROR_MSG, "The file %s couldn't be open for read. "
        "File %s, line %d", pStructSequence->szInputFilePath,
        __FILE__, __LINE__);

      return 0;
    }

    bInputFile = 1;
    break;

  /*The output file is not a must, but if the user wants to specify
    an output file, the file must be writable*/
  case 'O':
  case 'o':
    sscanf(optarg, "%s", pStructSequence->szOutputFilePath);

    if (strlen(pStructSequence->szOutputFilePath) == 0)
    {
      writeLog(WARNING_MSG, "No output data file. File %s, line %d. ",
        __FILE__, __LINE__);
    }
    else if(fopen(pStructSequence->szOutputFilePath, "w") == NULL)
    {
      writeLog(ERROR_MSG, "The file %s couldn't be open for write. "
        "File %s, line %d.", pStructSequence->szOutputFilePath,
        __FILE__, __LINE__);

      return 0;
    }
    break;

  /*The input for Gama value*/
  case 'G':
  case 'g':
     if (!sscanf(optarg, "%f", &(pStructSequence->fGama)))
     {
```

```
          writeLog(ERROR_MSG, "Could not get the input for Gama. "
            "File %s, line %d", __FILE__, __LINE__);

          return 0;
        }
        break;

    /*The input for Delta value*/
    case 'D':
    case 'd':
      if (!sscanf(optarg, "%d", &(pStructSequence->nDelta)))
      {
        writeLog(ERROR_MSG, "Could not get the input for delta. "
          "File %s, line %d", __FILE__, __LINE__);

         return 0;
       }
      break;

    /*The input for Delta value*/
    case 'M':
    case 'm':
      if (!sscanf(optarg, "%d", &(pStructSequence->nMaxGap)))
      {
        writeLog(ERROR_MSG, "Could not get the input for maxgap. "
          "File %s, line %d", __FILE__, __LINE__);

        return 0;
      }
      break;
    /*The input for Gama value*/

    case 'R':
    case 'r':
      if (!sscanf(optarg, "%f", &(pStructSequence->fDecRate)))
      {
        writeLog(ERROR_MSG, "Could not get the input for DecRate. "
          "File %s, line %d", __FILE__, __LINE__);

        return 0;
      }
      break;
    default:
      writeLog(WARNING_MSG, "Unsupprted command input and will be "
        "ignored. File %d, line %d", __FILE__, __LINE__);

      return 0;
  }
}

if(!bInputFile)
{
  writeLog(ERROR_MSG, "More arguments are expected. File %s, line %d",
```

```
        __FILE__, __LINE__);

      return 0;
    }

    return 1;
}

/*****************************************************************************
    Description:
      This function reads the data from the input file.
      The data format:
      Each line is a record of event that consist of a string for event and
      it is folloed by a numeric string that is the time of the event. Both
      are seperated by a space character.
      if the fist line starts with ">", then it is the comments for the
      sequence.
      After the function, we will get the event type list and MPO list for
      each event type.

    Input:
      pStructSequence: The structure will be populated

    Returned value:
      void

    History:
      03/07/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void ReadInputFile(PSEQUENCE pStructSequence)
{

    FILE *ptrFile                = NULL;
    char  szLine[MAX_LINE_LENGTH] = {0};
    char  szEventName[MAX_LINE_LENGTH] = {0};
    long  nTime          = 0;
    int   bDataStarted   = 0;
    long  nIndex         = 0;
    int   n              = -1;
    PLINKEDLIST pStructLinkedList = NULL;
    NODE        structNode        = {0};

    pStructLinkedList = initializeLinkedList(INT_TYPE);
    if(pStructLinkedList == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
        __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

        if ((ptrFile = fopen(pStructSequence->szInputFilePath, "r" )) == NULL)
        {
```

```
            writeLog(ERROR_MSG,  "Failed to open data file %s. File %s, line %d",
          pStructSequence->szInputFilePath, __FILE__, __LINE__);

        freeSpaceAndExit(pStructSequence, RETURN_ERROR);
        }

  writeLog(INFO_MSG, "Reading data from file %s. File %s, line %d",
     pStructSequence->szInputFilePath, __FILE__, __LINE__);

while ( fgets(szLine, MAX_LINE_LENGTH, ptrFile ) != NULL  )
{
     if (!bDataStarted && szLine[0] == '>')
     {
       pStructSequence->pszDescription =
         (char*)malloc((strlen(szLine)+1)*sizeof(char));

       if(pStructSequence->pszDescription == NULL)
       {
         writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
           __FILE__, __LINE__);

         freeSpaceAndExit(pStructSequence, RETURN_ERROR);
       }

       strcpy(pStructSequence->pszDescription, szLine+1);

       /*Only support 1 line of description*/
       bDataStarted = 1;
       continue;
     }

     if (bDataStarted && szLine[0] == '>')
     {
       writeLog(WARNING_MSG, "More than one lines of comments. "
         "File %s, line %d", __FILE__, __LINE__);

       continue;
     }

     /*Each line there would be only one event and its time stamp*/
     sscanf(szLine, "%s %ld", szEventName, &nTime);
     n = addEventType(pStructSequence, szEventName, "");
     if (n == -1)
     {
       writeLog(ERROR_MSG, "Failed to add the event type. File %s, line %d.",
         __FILE__, __LINE__);

       freeSpaceAndExit(pStructSequence, RETURN_ERROR);
     }
     else
     {

       /*Add the event to the event array*/
```

```
        addEventToArray(pStructSequence, nIndex, nTime, n);

        /*Prepare for the linked list*/
        pStructLinkedList = initializeLinkedList(INT_TYPE);
        if(pStructLinkedList == NULL)
        {
          writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
            __FILE__, __LINE__);

          freeSpaceAndExit(pStructSequence, RETURN_ERROR);
        }

        structNode.pContent = malloc(sizeof(int));
        *(int*)structNode.pContent = n;
        addNodeToListTail(pStructLinkedList, &structNode);
        free(structNode.pContent);

        addMPO(pStructSequence, &(pStructSequence->ppMPOList[0]),
          pStructLinkedList, nIndex, nIndex);

        destroyLinkedList(pStructLinkedList);
        nIndex++;
      }
}

fclose( ptrFile );
}

/*****************************************************************************
  Description:
    This function checks MPO list against Delta filter and remove the MPOs
    that do not satisfy the Delta requirement.

  Input:
    pStructSequence: The SQUENCE structure
    pMPOLinkedList: The linked list for MPO

  Returned value:
    void

  History:
    04/07/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void checkDeltaForMPOList(PSEQUENCE pStructSequence, PLINKEDLIST pMPOList)
{

  PNODE ptrMPONode       = NULL;
  PNODE ptrNextNode      = NULL;

  if(!pStructSequence)
  {

    writeLog(ERROR_MSG, "Invalid parameter. File %s, line %d",
```

```
      __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  if(!(pMPOList))
  {

    writeLog(ERROR_MSG, "Invalid parameter. File %s, line %d",
      __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  ptrMPONode =  pMPOList->pListNode;

  for (; ptrMPONode; ptrMPONode = ptrNextNode)
  {
    ptrNextNode = ptrMPONode->pNextNode;
    checkDeltaForAMPO(pStructSequence, pMPOList, ptrMPONode);
  }
}

/*****************************************************************************
  Description:
    This function checks a MPO against minimum support (delta) requirement
    and remove the MPO if it does not satisfy the requirement.
    Make sure the pMPONode is a node of pMPOLinkedList before calling this
    function.

  Input:
    pStructSequence: The SQUENCE structure
    pMPOLinkedList:  The linked list for MPO
    pStructMPO:      The MPO node that needs to be checked

  Returned value:
    1 if the MPO node was removed from the linked list due to fewer
    occurrence than the Delta requirement.
    0 otherwise.

  History:
    04/11/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int checkDeltaForAMPO(PSEQUENCE pStructSequence, PLINKEDLIST pMPOList,
                      PNODE ptrMPONode)
{

  PNODE ptrMPOClusterNode = NULL;
  PMPOCLUSTER pStructMPOCluster = NULL;
  PLINKEDLIST ptrMPOClusterList = NULL;
  PLINKEDLIST ptrEndIngIndexList= NULL;
  long  nOccurrence       = 0;
```

```
  if(!pStructSequence)
  {
    writeLog(ERROR_MSG, "Invalid parameter. File %s, line %d",
      __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  if(!pMPOList)
  {
    writeLog(ERROR_MSG, "Invalid parameter. File %s, line %d",
      __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  ptrMPOClusterList = ((PMPO)(ptrMPONode->pContent))->pMPOClusterList;

  /*There no occurrence for the episode, remove it*/
  if (!ptrMPOClusterList)
  {
    removeNodefromList(pMPOList, ptrMPONode);
    return 1;
  }

  /*If the number of the MO occurrence is less than the minimum support
  requirement, then remove it. Support is defined on the MO, not MPO,
  so we need only count on the MPO cluster numbers.*/
  if(ptrMPOClusterList->nCount < pStructSequence->nDelta)
  {
    removeNodefromList(pMPOList, ptrMPONode);
    return 1;
  }
  else
    return 0;
}

/*****************************************************************************
  Description:
    This function form a MPOCLSURT node for linked list. Don't forget to
    free the memory since dynamic allocation is used.

  Input:
    pStructSequence: The SEQUENCE structure
    nStartIndex:     The start index in the event sequence.

  Returned value:
    NULL if failed, valid pointer if sucessful

  History:
    04/25/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
```

```
PNODE formAnMPOClusterNode(PSEQUENCE pStructSequence, long nStartIndex)
{
  MPOCLUSTER structMPOCluster = {0};
  PNODE      pTempNode        = NULL;

        structMPOCluster.nStartPosInEventList = nStartIndex;
  structMPOCluster.pEndingPosInEventList = initializeLinkedList(INT_TYPE);
  if(structMPOCluster.pEndingPosInEventList == NULL)
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
      __FILE__, __LINE__);

    return NULL;
  }

  /*Form a MPO cluster node*/
  pTempNode = makeANodeFromValue(MPOCLUSTER_TYPE,(void*)&structMPOCluster);
  destroyLinkedList(structMPOCluster.pEndingPosInEventList);
  if(pTempNode == NULL)
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
      __FILE__, __LINE__);

    return NULL;
  }

  return pTempNode;
}

/*****************************************************************************
  Description:
    This function add a MPO into the MPO linked list

  Input:
    pStructSequence: The structure will be populated

  Returned value:
    0 failed, 1 sucessful

  History:
    03/07/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int addMPO(PSEQUENCE pStructSequence, PLINKEDLIST *ppMPOList,
        PLINKEDLIST pEventLinkedList, int nStartIndex,  int nEndIndex)
{
  PLINKEDLIST ptrEventLinkedList   = NULL;
  PLINKEDLIST pMPOClusterList      = NULL;
  PNODE       pMPOClusterNode      = NULL;
  PNODE       pMPONode             = NULL;
  PNODE       pIntNode             = NULL;
  PNODE       pTempNode            = NULL;
  int         bFound               = 0;
  int         nReturn              = 0;
```

```
PMPOCLUSTER pStructMPOCluster    = NULL;
MPO         structMPO            = {0};
PMPO        pStructMPO           = NULL;

/*Create the node for the futire use. Don't forget to free it
before returning value*/
pIntNode = makeANodeFromValue(INT_TYPE, (void*)&nEndIndex);

if(pIntNode == NULL)
{
  writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
    __FILE__, __LINE__);

  freeSpaceAndExit(pStructSequence, RETURN_ERROR);
}

/*MPO list did not exist. Do everything*/
if(!(*ppMPOList))
{
  *ppMPOList = initializeLinkedList(MPO_TYPE);
  if(*ppMPOList == NULL)
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
      __FILE__, __LINE__);

    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  structMPO.pMPOClusterList = initializeLinkedList(MPOCLUSTER_TYPE);
  if(structMPO.pMPOClusterList == NULL)
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d.",
      __FILE__, __LINE__);

    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  /*Don't bother allocate the space here*/
  structMPO.pEpisodeEventList = NULL;
  structMPO.nTotalOccurrence = 0;
  structMPO.nFLM = 0;

  nReturn = addNodeToListTail(*ppMPOList, (PNODE)&structMPO);
  destroyLinkedList(structMPO.pMPOClusterList);

  pStructMPO = (PMPO)((*ppMPOList)->pLastNode->pContent);

  /*add the episode event list*/
  pStructMPO->pEpisodeEventList = duplicateLinkedList(pEventLinkedList);
  if(pStructMPO->pEpisodeEventList == NULL)
  {
```

```
    writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
      __FILE__, __LINE__);

    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  /*add the new MPO cluster*/
  pMPOClusterList = pStructMPO->pMPOClusterList;

  /*make an MPOCLUSTER node for the linked list*/
  pTempNode = formAnMPOClusterNode(pStructSequence,nStartIndex);
  /*Error message was printed out in the formAnMPOClusterNode function*/
  if (pTempNode == NULL)
  {
    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  nReturn = addNodeToListTail(pMPOClusterList, pTempNode);
  freeOneNode(MPOCLUSTER_TYPE, pTempNode);

  if(nReturn == 0)
  {
    writeLog(ERROR_MSG, "Failed to add MPO cluster node. "
      "File %s, line %d.", __FILE__, __LINE__);

    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  pStructMPOCluster
    = (PMPOCLUSTER)(pMPOClusterList->pLastNode->pContent);
  ptrEventLinkedList = pStructMPOCluster->pEndingPosInEventList;
  nReturn = addNodeToListTail(ptrEventLinkedList,pIntNode);
  pStructMPO->nTotalOccurrence++;
  freeOneNode(INT_TYPE, pIntNode);

  if(nReturn == 0)
  {
    writeLog(ERROR_MSG, "Failed to add ending position node. "
      "File %s, line %d.", __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }
}

/*MPO list existed, search for the episode in the pEpisodeEventList*/
pMPONode = (*ppMPOList)->pListNode;
for(; pMPONode; pMPONode = pMPONode->pNextNode)
{
  pStructMPO = (PMPO)(pMPONode->pContent);
```

```
  if(equalList(pStructMPO->pEpisodeEventList, pEventLinkedList))
  {
    bFound = 1;
    break;
  }
}

/*Found the Episode*/
if(bFound)
{
  pMPOClusterList = ((PMPO)pMPONode->pContent)->pMPOClusterList;

  /*The MPO cluster list does not exist. Then create the MPOLusterList,
  MPO cluster and add the nEndIndex*/
  if(!pMPOClusterList)
  {
    pStructMPO->pMPOClusterList = initializeLinkedList(MPOCLUSTER_TYPE);
    if(pStructMPO->pMPOClusterList == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
        __FILE__, __LINE__);

      freeOneNode(INT_TYPE, pIntNode);
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    /*Make an MPOCLUSTER node for linked list*/
    pTempNode = formAnMPOClusterNode(pStructSequence,nStartIndex);
    /*Error message was printed out in the formAnMPOClusterNode function*/
    if (pTempNode == NULL)
    {
      freeOneNode(INT_TYPE, pIntNode);
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    nReturn = addNodeToListTail(pMPOClusterList,pTempNode);
    freeOneNode(MPOCLUSTER_TYPE, pTempNode);
    if(nReturn == 0)
    {
      writeLog(ERROR_MSG, "Failed to add MPO cluster node. "
        "File %s, line %d.", __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    pStructMPOCluster
      = (PMPOCLUSTER)(pMPOClusterList->pLastNode->pContent);
    ptrEventLinkedList = pStructMPOCluster->pEndingPosInEventList;
    nReturn = addNodeToListTail(ptrEventLinkedList,pIntNode);
    pStructMPO->nTotalOccurrence++;
    freeOneNode(INT_TYPE, pIntNode);

    if(nReturn == 0)
```

```
  {
    writeLog(ERROR_MSG, "Failed to add ending position node. "
      "File %s, line %d.", __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  return nReturn;
}

/*MPO cluster list exists, then search for the cluster that has the
same start point*/
bFound = 0;    /*The flag will be reused again in the new search*/

pMPOClusterNode = pMPOClusterList->pListNode;

for (; pMPOClusterNode; pMPOClusterNode = pMPOClusterNode->pNextNode)
{
  pStructMPOCluster = (PMPOCLUSTER)(pMPOClusterNode->pContent);

  if( pStructMPOCluster->nStartPosInEventList == nStartIndex)
  {
    bFound = 1;
    break;
  }
}

/*If the MPO cluster was found, then add the new the node to the
pEndingPosInEventList if the nEndIndex is not in it, otherwise do
nothing*/
if(bFound)
{
  ptrEventLinkedList = pStructMPOCluster->pEndingPosInEventList;

  /*If the ending position linked list does not exist, create it*/
  if(!ptrEventLinkedList)
  {
    ptrEventLinkedList = initializeLinkedList(INT_TYPE);
    if(ptrEventLinkedList == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
        __FILE__, __LINE__);

      freeOneNode(INT_TYPE, pIntNode);
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    pStructMPOCluster->pEndingPosInEventList = ptrEventLinkedList;
    nReturn = addNodeToListTail(ptrEventLinkedList, pIntNode);
    pStructMPO->nTotalOccurrence++;
  }
  /*If the ending position linked list exists, search the node
  if found, then give a warning, otherwise, add the node it into it*/
```

```
  else
  {
    searchNodeInList(ptrEventLinkedList,pIntNode, &pTempNode);

    if(!pTempNode)
    {
      nReturn = addNodeToListTail(ptrEventLinkedList, pIntNode);
      pStructMPO->nTotalOccurrence++;
    }
    else
      nReturn = 1;
    }
  }

  freeOneNode(INT_TYPE, pIntNode);
  return nReturn;
}

/*The the MPO cluster was not found, then create the new MPO cluster and
add the new nEndIndex to the pEndingPosInEventList*/
else
{
  /*Make an MPOCLUSTRE node for linked list*/
  pTempNode = formAnMPOClusterNode(pStructSequence,nStartIndex);
  /*Error message was printed out in the formAnMPOClusterNode */
  if (pTempNode == NULL)
  {
    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  nReturn = addNodeToListTail(pMPOClusterList,pTempNode);
  freeOneNode(MPOCLUSTER_TYPE, pTempNode);

  if(nReturn == 0)
  {
    writeLog(ERROR_MSG, "Failed to add MPO cluster Node. "
      "File %s, line %d. ", __FILE__, __LINE__);

    freeOneNode(INT_TYPE, pIntNode);
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  pStructMPOCluster = (PMPOCLUSTER)(pMPOClusterList->pLastNode->pContent);
  nReturn = addNodeToListTail(
            pStructMPOCluster->pEndingPosInEventList,pIntNode);
  pStructMPO->nTotalOccurrence++;

  freeOneNode(INT_TYPE, pIntNode);

  if(nReturn == 0)
  {
    writeLog(ERROR_MSG, "Failed to add the ending position Node. "
```

```
          "File %s, line %d. ", __FILE__, __LINE__);
        freeSpaceAndExit(pStructSequence, RETURN_ERROR);
      }
      return nReturn;
    }
  }
  /*Did not found the Episode, then create the MPO structure for the new
  episode and then create and add the MPO cluster and add the nEndIndex */
  else
  {
    structMPO.pMPOClusterList = initializeLinkedList(MPOCLUSTER_TYPE);
    if((structMPO.pMPOClusterList) == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
        __FILE__, __LINE__);

      freeOneNode(INT_TYPE, pIntNode);
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    /*Don't bother allocate the space here*/
    structMPO.pEpisodeEventList = NULL;
    structMPO.nFLM  = 0;
    structMPO.nTotalOccurrence = 0;
    pTempNode = makeANodeFromValue(MPO_TYPE,(void*)&structMPO);
    if(pTempNode == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
        __FILE__, __LINE__);

      freeOneNode(INT_TYPE, pIntNode);
      destroyLinkedList(structMPO.pMPOClusterList);
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    nReturn = addNodeToListTail(*ppMPOList, pTempNode);
    destroyLinkedList(structMPO.pMPOClusterList);
    destroyLinkedList(structMPO.pEpisodeEventList);
    freeOneNode(MPO_TYPE, pTempNode);

    if(nReturn == 0)
    {
      writeLog(ERROR_MSG, "Failed to add the MPO node. File %s, line %d. ",
          __FILE__, __LINE__);

      freeOneNode(INT_TYPE, pIntNode);
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    /*add the MPO cluster to the list. Form a node, add it*/
    pMPOClusterList
      =((PMPO)((*ppMPOList)->pLastNode->pContent))->pMPOClusterList;
```

```
/*Make an MPOCLUSTER node for linked list*/
pTempNode = formAnMPOClusterNode(pStructSequence,nStartIndex);
/*Error message was printed out in the formAnMPOClusterNode function*/
if (pTempNode == NULL)
{
  freeOneNode(INT_TYPE, pIntNode);
  freeSpaceAndExit(pStructSequence, RETURN_ERROR);
}

nReturn = addNodeToListTail(pMPOClusterList, pTempNode);
freeOneNode(MPOCLUSTER_TYPE, pTempNode);

if(nReturn == 0)
{
  writeLog(ERROR_MSG, "Failed to add ending position node. "
    "File %s, line %d.", __FILE__, __LINE__);

  freeOneNode(INT_TYPE, pIntNode);
  freeSpaceAndExit(pStructSequence, RETURN_ERROR);
}

pStructMPOCluster = (PMPOCLUSTER)(pMPOClusterList->pLastNode->pContent);

pStructMPOCluster->pEndingPosInEventList = initializeLinkedList(INT_TYPE);
if(pStructMPOCluster->pEndingPosInEventList == NULL)
{
  writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
    __FILE__, __LINE__);

  freeOneNode(INT_TYPE, pIntNode);
  freeSpaceAndExit(pStructSequence, RETURN_ERROR);
}

nReturn = addNodeToListTail(
          pStructMPOCluster->pEndingPosInEventList,pIntNode);
freeOneNode(INT_TYPE, pIntNode);
if(nReturn == 0)
{
  writeLog(ERROR_MSG, "Failed to add the ending positon node. "
    "File %s, line %d.",  __FILE__, __LINE__);

  freeSpaceAndExit(pStructSequence, RETURN_ERROR);
}

pStructMPO = (PMPO)((*ppMPOList)->pLastNode->pContent);
pStructMPO->nTotalOccurrence++;

/*add the episode event list to the structure*/
ptrEventLinkedList = duplicateLinkedList(pEventLinkedList);
if(ptrEventLinkedList == NULL)
{
  writeLog(ERROR_MSG, "Memory error. File %s, line %d. ",
    __FILE__, __LINE__);
```

```
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    ((PMPO)((*ppMPOList)->pLastNode->pContent))->pEpisodeEventList
      = ptrEventLinkedList;

    return nReturn;
  }
}

/*****************************************************************************
  Description:
    This function adds an event to the event array

 Input:
    pStructSequence:  The SEQUENCE structure

  Returned value:
    void

  History:
    04/23/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void addEventToArray(PSEQUENCE pStructSequence, long nIndex,
           long nTime, int nTypeIndex)
{
  PEVENT pStructEvent = NULL;
  long   lNewArraySize = 0;

  if(pStructSequence->pEventArray == NULL)
  {
    pStructSequence->pEventArray
      = (PEVENT)malloc(DEFAULT_EVENT_SIZE*sizeof(EVENT));

    if(pStructSequence->pEventArray == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }

    pStructSequence->lEventArraySize = DEFAULT_EVENT_SIZE;
    pStructSequence->lEventNumber = 0;
  }

  pStructSequence->lEventNumber++;

  if(pStructSequence->lEventNumber> pStructSequence->lEventArraySize)
  {
    pStructSequence->lEventArraySize += DEFAULT_EVENT_SIZE;
    pStructSequence->pEventArray =(PEVENT)realloc(
```

```
      pStructSequence->pEventArray, (pStructSequence->lEventArraySize)*sizeof(EVENT));

    if(pStructSequence->pEventArray == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }
  }

  pStructEvent
    = pStructSequence->pEventArray + (pStructSequence->lEventNumber-1);

  pStructEvent->nPosInEventList = nIndex;
  pStructEvent->nTime = nTime;
  pStructEvent->nPosInEventTypeList = nTypeIndex;
}

/*****************************************************************************
  Description:
    This function adds the event type to the event type linked list

  Input:
    pStructSequence:  The SEQUENCE structure
    pszEventType:     The event type name to be added
    pszDesc:          The descritption of the event type.

  Returned value:
    int: The index in the linked list (ncount -1)

  History:
    03/08/2007: Created by Chuanwu (Steven) Xiong
    04/23/2007: Changed the event type list to array
*****************************************************************************/
int addEventType(PSEQUENCE pStructSequence,char* pszEventType, char* pszDesc)
{
  int nIndex  = 0;
  PEVENT_TYPE pStructEventType = NULL;
  char* pszTemp     = NULL;

  if(pStructSequence->pEventTypeArray == NULL)
  {
    pStructSequence->pEventTypeArray
      = (PEVENT_TYPE)malloc(DEFAULT_EVENT_TYPE_NUMBER*sizeof(EVENT));

    if(pStructSequence->pEventTypeArray == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }
```

```
    pStructSequence->nEventTypeArraySize = DEFAULT_EVENT_TYPE_NUMBER;
    pStructSequence->nEventTypeNumber = 0;
  }

  nIndex = searchEventType(pStructSequence, pszEventType);

  if(nIndex != -1)
    return nIndex;

  pStructSequence->nEventTypeNumber++;

  if(pStructSequence->nEventTypeNumber > pStructSequence->nEventTypeArraySize)
  {
    pStructSequence->nEventTypeArraySize += DEFAULT_EVENT_TYPE_NUMBER;
    pStructSequence->pEventTypeArray =(PEVENT_TYPE)realloc(
      pStructSequence->pEventTypeArray, (pStructSequence->nEventTypeArraySize)*
       sizeof(EVENT_TYPE));

    if(pStructSequence->pEventTypeArray == NULL)
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);

      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
    }
  }

  pStructEventType
    = pStructSequence->pEventTypeArray + (pStructSequence->nEventTypeNumber-1);

  pStructEventType->pszEvent
    = (char*)malloc(sizeof(char)*(strlen(pszEventType)+1));

  pStructEventType->pszDescription
    = (char*)malloc(sizeof(char)*(strlen(pszDesc)+1));

  if((pStructEventType->pszDescription == NULL) ||
    (pStructEventType->pszEvent == NULL))
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);

    freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  strcpy(pStructEventType->pszEvent, pszEventType);
  strcpy(pStructEventType->pszDescription, pszDesc);

  return pStructSequence->nEventTypeNumber - 1;
}
```

```
/*****************************************************************************
  Description:
    This function searches the event type in the pEventTypeList and returns
    the index in the linked list and the pointer to the node

  Input:
    pStructSequence: The SEQUENCE structure
    pszEventType: The event type name to be searched
    pNode: The pointer to the node if the event type is found.

  Returned value:
    int: The index in the linked list. -1 means not found

  History:
    03/08/2007: Created by Chuanwu (Steven) Xiong
    04/23/2007; Changed from event type list to event type array
*****************************************************************************/
int searchEventType(PSEQUENCE pStructSequence, char * pszEventType)
{
  int bFound = 0;
  int i = 0;
  PEVENT_TYPE pStructEventType = NULL;

  ASSERT(pStructSequence);

  for(i = 0; i < pStructSequence->nEventTypeNumber; i++)
  {
    pStructEventType = pStructSequence->pEventTypeArray + i;
    if(stricmp(pszEventType, pStructEventType->pszEvent) == 0)
    {
      bFound = 1;
      break;
    }
  }

  if(bFound)
    return i;
  else
    return -1;
}

/*****************************************************************************
  Description:
    This function prints MPO information from a MPO node in the MPO List

 Input:
    pStructSequence:  The SEQUENCE structure
    ptrMPONode        MPO node that needs to be printed out

  Returned value:
    void

  History:
```

```
    04/11/2007: Created by Chuanwu (Steven) Xiong
*************************************************************************/
void printMPO(PSEQUENCE pStructSequence, PNODE ptrMPONode)
{
  FILE  *ptrFile    = NULL;
  PMPO  pStructMPO  = NULL;
  PNODE ptrMPOCLuster     = NULL;
  PNODE ptrMPOClusterNode = NULL;
  PNODE ptrEventNode      = NULL;
  PEVENT pstructEvent     = NULL;
  PNODE ptrNode           = NULL;
  PEVENT_TYPE pStructEventType  = NULL;
  PMPOCLUSTER pStructMPOCluster = NULL;
  int   nIndex   = 0;
  int   nStartIndex = 0,  nEndingIndex = 0;
  int   nStartTime = 0,   nEndingTime = 0;

  if(ptrMPONode == NULL || ptrMPONode->pContent == NULL)
    return;

  pStructMPO = (PMPO)(ptrMPONode->pContent);

  if (pStructMPO->nTotalOccurrence == 0)
    return;

  ptrFile = fopen(pStructSequence->szOutputFilePath, "a");

  /*Print the episode sequence*/
  fprintf(ptrFile, "\n\nThe MPO occurences for Episode Sequence: ");
  ptrEventNode = pStructMPO->pEpisodeEventList->pListNode;
  for (; ptrEventNode; ptrEventNode = ptrEventNode->pNextNode)
  {
    nIndex = ((PEVENT)(ptrEventNode->pContent))->nPosInEventTypeList;

    pStructEventType = pStructSequence->pEventTypeArray + nIndex;

     if(ptrEventNode->pNextNode)
       fprintf(ptrFile, "%s->", pStructEventType->pszEvent);
    else /*the last event, arrow mark doe not needed*/
       fprintf(ptrFile, "%s\n", pStructEventType->pszEvent);
  }

  fprintf(ptrFile, "%-10s%-10s%-10s%-10s%-10s%s\n", "S_Index", "E_Index",
    "W_Index ", "S_Time", "E_Time", "W_Time");
  /*Print out the start index, ending index and thw width information
  for every mpo occurrence*/
  ptrMPOClusterNode = pStructMPO->pMPOClusterList->pListNode;
  for (; ptrMPOClusterNode; ptrMPOClusterNode= ptrMPOClusterNode->pNextNode)
  {
    pStructMPOCluster =  ((PMPOCLUSTER)(ptrMPOClusterNode->pContent));

    /*If the cluster is NULL or the ending pistion list is empty, do
    nothing*/
```

```
    if(!pStructMPOCluster)
      continue;
    if (!(pStructMPOCluster->pEndingPosInEventList))
      continue;

    /*Get the information for the start event in the episode*/
    nStartIndex
      = ((PMPOCLUSTER)(ptrMPOClusterNode->pContent))->nStartPosInEventList;
    pstructEvent = pStructSequence->pEventArray + nStartIndex;

    nStartTime = pstructEvent->nTime;

    /*Get the information for the ending event*/
    ptrNode =pStructMPOCluster->pEndingPosInEventList->pListNode;
    for (; ptrNode; ptrNode = ptrNode->pNextNode)
    {
      nEndingIndex= *((int*)(ptrNode->pContent));

      pstructEvent = pStructSequence->pEventArray + nEndingIndex;

      nEndingTime = pstructEvent->nTime;

      fprintf(ptrFile, "%-10d%-10d%-10d%-10d%-10d%-10d\n", nStartIndex, nEndingIndex,
        nEndingIndex-nStartIndex, nStartTime, nEndingTime,
        nEndingTime-nStartTime);
    }
  }

  fprintf(ptrFile, "-----End for the episode------\n\n\n");
  fclose(ptrFile);
}


/*****************************************************************************
  Description:
    This function prints FLM information from a MPO node in the MPO List

 Input:
    pStructSequence:  The SEQUENCE structure
    ptrMPONode        MPO node that needs to be printed out

  Returned value:
    void

  History:
    05/08/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void printFLM(PSEQUENCE pStructSequence)
{
  FILE  *ptrFile    = NULL;
  PMPO  pStructMPO  = NULL;
  PNODE ptrEventNode = NULL;
  PNODE pMPONode    = NULL;
```

```
  PEVENT_TYPE pStructEventType = NULL;
  int   nIndex     = 0;
  int   i          = 0;
  int   j          = 0;

  ptrFile = fopen(pStructSequence->szOutputFilePath, "a");

  for (i = 0; i < pStructSequence->nMPOListNumber; i++)
  {
    if(pStructSequence->ppMPOList[i]->nCount == 0)
      break;

    j = 1;

    pMPONode = pStructSequence->ppMPOList[i]->pListNode;
    for(; pMPONode; pMPONode = pMPONode->pNextNode)
    {
      if(pMPONode == NULL || pMPONode->pContent == NULL)
        break;

      pStructMPO = (PMPO)(pMPONode->pContent);

      if (pStructMPO->nTotalOccurrence == 0)
        break;

      if(pStructMPO->nFLM > 0)
      {
        /*Print the episode sequence*/
        fprintf(ptrFile, "%-3d:%-3dThe FLM for Episode Sequence \" ", i, j++);
        ptrEventNode = pStructMPO->pEpisodeEventList->pListNode;
        for (; ptrEventNode; ptrEventNode = ptrEventNode->pNextNode)
        {
          nIndex = ((PEVENT)(ptrEventNode->pContent))->nPosInEventTypeList;

          pStructEventType = pStructSequence->pEventTypeArray + nIndex;

           if(ptrEventNode->pNextNode)
             fprintf(ptrFile, "%s->", pStructEventType->pszEvent);
          else /*the last event, arrow mark doe not needed*/
          {
               fprintf(ptrFile, "%s\" is %d\n\n", pStructEventType->pszEvent,
                 pStructMPO->nFLM);
          }
        }
      }
    }
  }
  fclose(ptrFile);
}


/*****************************************************************************
  Description:
```

```
    This function free the space allocated for the members in the SEQUENCE
    structure.

  Input:
    pStructSequence:  The SEQUENCE structure
    nReturnType:      The return type

  Returned value:
    void

  History:
    04/23/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void freeSpaceAndExit(PSEQUENCE pStructSequence, RETURN_TYPE nReturnType)
{
  int i = 0;

  /*We need only free the space for the dunamically allocated members in
  the structure. The structure itself is not dynamically allcoated. */
  if (pStructSequence)
  {
    if(pStructSequence->pEventArray)
      free(pStructSequence->pEventArray);

    if(pStructSequence->pEventTypeArray)
      free(pStructSequence->pEventTypeArray);

    if(pStructSequence->pszDescription)
      free(pStructSequence->pszDescription);
  }

  /*free the MPOList array*/
  if(pStructSequence->ppMPOList)
  {
    for (i = 0; i < pStructSequence->nMPOListNumber; i++)
    {
      destroyLinkedList(pStructSequence->ppMPOList[i]);
    }

    free(pStructSequence->ppMPOList);
  }

  if ((nReturnType == RETURN_ERROR) || (nReturnType == RETURN_WARNING))
    exit(1);

  if(nReturnType == RETURN_SUCCESS)
    exit(0);
}

/*****************************************************************************
  Description:
    This function prints some information in  log file
```

```
  Input:
    pStructSequence:  The SEQUENCE structure

  Returned value:
    void

  History:
    04/23/2007: Created by Chuanwu (Steven) Xiong
*************************************************************************/
void printLogFileHeader(PSEQUENCE pStructSequence)
{
  char     szBuffer[1000] = {0};
  char     szLogFileName[256]= {0};
  FILE     *ptrLogFile       = NULL;
  struct tm *pStructTime     = NULL;
  long     ltime             = 0;
  int      nResult       = 0;
#ifdef   WIN32
  struct _stat buf  = {0};
  WORD         wVersionRequested   = {0};
  WSADATA      wsaData = {0};
#else
  struct stat buf   = {0};
#endif

  /*Check if the log folder is there. Create it if not.*/
#ifdef WIN32
  sprintf(szLogFileName, "log\\%s%d.log", LOG_FILE, getpid());
  nResult = _stat( "log", &buf );
#else
  sprintf(szLogFileName, "log/%s%d.log", LOG_FILE, getpid());
  nResult = stat("log", &buf);
#endif

  /*Create the log folder*/
  if(nResult != 0)
  {
    nResult = system("mkdir log");

   /*If cannot create the log folder, exit*/
    if(nResult != 0)
      freeSpaceAndExit(pStructSequence, RETURN_ERROR);
  }

  /*Open can clear the exiting content in the file*/
  ptrLogFile = fopen(szLogFileName, "w");

  if(ptrLogFile == NULL)
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);

  fprintf(ptrLogFile, "%20s%s\n", " ", "The Log For Episode\n\n");

  fprintf(ptrLogFile, "%15sProcess ID: %d\n", " ", getpid());
```

```
  #ifdef   WIN32
  wVersionRequested = MAKEWORD (1,1);
  if((WSAStartup(wVersionRequested, &wsaData)) == 0)
  {
  #endif

  gethostname(szBuffer, 1000);

  #ifdef   WIN32
  }
  WSACleanup();
  #endif

  fprintf(ptrLogFile, "%15sHost Name: %s\n", " ", szBuffer);

  time( &ltime );                  /*Get the time*/
  pStructTime = localtime( &ltime );   /*Convert it to the tm structure*/
  if( pStructTime->tm_hour > 12 )      /* Set up extension. */
    strcpy( szBuffer, "PM" );
  else
    strcpy( szBuffer, "AM" );

  if( pStructTime->tm_hour > 12 )      /* Convert from 24-hour */
    pStructTime->tm_hour -= 12;       /*   to 12-hour clock.  */
  if( pStructTime->tm_hour == 0 )      /*Set hour to 12 if midnight. */
    pStructTime->tm_hour = 12;

  fprintf(ptrLogFile,  "%15sTime: %.19s %s\n\n\n", " ", asctime(pStructTime), szBuffer);

  fclose(ptrLogFile);
}

/*****************************************************************************
  Description:
    This function prints some information in  output file file

 Input:
    pStructSequence:  The SEQUENCE structure

  Returned value:
    void

  History:
    04/23/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void printOutputFileHeader(PSEQUENCE pStructSequence)
{
  char     szBuffer[1000] = {0};
  FILE     *ptrOutputFile    = NULL;
  struct tm *pStructTime      = NULL;
  long     ltime             = 0;
#ifdef   WIN32
```

```
  struct _stat buf  = {0};
  WORD        wVersionRequested   = {0};
  WSADATA      wsaData = {0};
#else
  struct stat buf   = {0};
#endif

  /*Open can clear the exiting content in the file*/
  ptrOutputFile = fopen(pStructSequence->szOutputFilePath, "w");

  if(ptrOutputFile == NULL)
    freeSpaceAndExit(pStructSequence, RETURN_ERROR);

  fprintf(ptrOutputFile, "%20s%15s\n", " ", "The Result For Episode Analysis\n\n");

  fprintf(ptrOutputFile, "%15s%-22s%d\n", " ", "Process ID: ", getpid());

  #ifdef WIN32
    wVersionRequested = MAKEWORD (1,1);
    if((WSAStartup(wVersionRequested, &wsaData)) == 0)
    {
  #endif

       gethostname(szBuffer, 1000);

  #ifdef WIN32
    }
    WSACleanup();
  #endif

  fprintf(ptrOutputFile, "%15s%-22s%s\n", " ", "Host Name: ", szBuffer);

  time( &ltime );                    /*Get the time*/
  pStructTime = localtime( &ltime );   /*Convert it to the tm structure*/
  if( pStructTime->tm_hour > 12 )      /* Set up extension. */
    strcpy( szBuffer, "PM" );
  else
    strcpy( szBuffer, "AM" );

  if( pStructTime->tm_hour > 12 )      /* Convert from 24-hour */
    pStructTime->tm_hour -= 12;       /*   to 12-hour clock.  */
  if( pStructTime->tm_hour == 0 )      /*Set hour to 12 if midnight. */
    pStructTime->tm_hour = 12;

  fprintf(ptrOutputFile,  "%15s%-22s%.19s %s\n", " ", "Time: ",
    asctime(pStructTime), szBuffer);

  fprintf(ptrOutputFile,  "%15s%-22s%s\n", " ", "Input File: ",
    pStructSequence->szInputFilePath);

  fprintf(ptrOutputFile,  "%15s%-22s%d\n", " ", "MaxGap: ",
    pStructSequence->nMaxGap);
```

```
  fprintf(ptrOutputFile,  "%15s%-22s%d\n", " ", "Minimum Support: ",
    pStructSequence->nDelta);

  fprintf(ptrOutputFile,  "%15s%-22s%4.3f\n", " ", "Minimum Confidence: ",
    pStructSequence->fGama);

  fprintf(ptrOutputFile,  "%15s%-22s%4.3f\n", " ", "DecRate: ",
    pStructSequence->fDecRate);

  fprintf(ptrOutputFile,  "%15s%-22s%ld\n", " ", "Total Events: ",
    pStructSequence->lEventNumber);

  fprintf(ptrOutputFile,  "%15s%-22s%d\n", " ", "Total Event Type: ",
    pStructSequence->nEventTypeNumber);

  if(pStructSequence->pszDescription)
  {
    fprintf(ptrOutputFile,  "%15s%s\n", " ", "Description: ",
      pStructSequence->pszDescription);
  }

  fclose(ptrOutputFile);
}

/*****************************************************************************
  Description:
    This function prints out the command format for running this application
  the command line can have the foollowing options. 1 and 2 are required, the
  others are optional.
    1. The command executable
    2. The input file that store the event sequence. after -I or -i
    3. The threshhold for delta, after -D or -d
    4. The threshhold for gama, after -G or -g
    5. The value for the maxgap, after -M or -m
    6. The output file name, after -O or -o

  Input:
    void

  Returned value:
    void

  History:
    05/04/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void printCommandFormat(void)
{
  printf("The command line can have the foollowing options:\n");
  printf("1. The command executable\n");
  printf("2. The input file that store the event sequence. after -I or -i\n");
  printf("3. The threshhold for delta, after -D or -d\n");
  printf("4. The threshhold for gama, after -G or -g\n");
  printf("5. The value for the maxgap, after -M or -m\n");
```

```
  printf("6. The output file name, after -O or -o\n");
  printf("The option 1 and 2 are required. The others are optional, but\n");
  printf("the default value will be taken.\n\n");
  printf("The example of the command:\n");
  printf("episode -i test.txt -G 0.1 -d 1\n");

}



/*****************************************************************************
  Description:
    This function prints MPO information from a MPO node in the MPO List

 Input:
    pStructSequence:  The SEQUENCE structure
    ptrMPONode        MPO node that needs to be printed out

  Returned value:
    void

  History:
    04/11/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void printEpisode(PSEQUENCE pStructSequence, PNODE ptrMPONode)
{
  PMPO  pStructMPO  = NULL;
  PNODE ptrEventNode      = NULL;
  PEVENT_TYPE pStructEventType = NULL;
  char szBuffer[1000] = {0};
  int   nIndex        = 0;

  if(ptrMPONode == NULL || ptrMPONode->pContent == NULL)
    return;

  pStructMPO = (PMPO)(ptrMPONode->pContent);

  if (pStructMPO->nTotalOccurrence == 0)
    return;

  /*Print the episode sequence*/
  strcpy(szBuffer, "The Episode Sequence: ");
  ptrEventNode = pStructMPO->pEpisodeEventList->pListNode;
  for (; ptrEventNode; ptrEventNode = ptrEventNode->pNextNode)
  {
    nIndex = ((PEVENT)(ptrEventNode->pContent))->nPosInEventTypeList;

    pStructEventType = pStructSequence->pEventTypeArray + nIndex;

     if(ptrEventNode->pNextNode)
     {
       strcat(szBuffer, pStructEventType->pszEvent);
```

```
        strcat(szBuffer, "->");
      }
     else /*the last event, do not need arrow mark*/
        strcat(szBuffer, pStructEventType->pszEvent);
  }
  writeLog(INFO_MSG, "%s", szBuffer);

}
```

linkedList.h

```
#ifndef _LINKEDLIST_H
#define _LINKEDLIST_H

/*****************************************************************************
                Macro define section
*****************************************************************************/



/*****************************************************************************
                    include section
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include "log.h"


/*****************************************************************************
            Struction definition section
*****************************************************************************/

/* The enumeration is the type information for the data that are  stored in
   the linked list. User could add more.
 */
typedef enum
{
        INT_TYPE,                       /*Integer type*/
  UNSIGNED_INT_TYPE,      /*unsigned integer type*/
  LONG_TYPE,                    /*Long type*/
  UNSIGNED_LONG_TYPE,  /*unsigned LONG type*/
  SHORT_TYPE,                   /*short type*/
  UNSIGNED_SHORT_TYPE, /*unsigned short type*/
        FLOAT_TYPE,             /*float type*/
        DOUBLE_TYPE,            /*double type*/
  CHAR_Type,                    /*character type*/
        STRING_TYPE,                /*character string type*/
  /*STRUCTURE_TYPE    user defined structure type*/
  EVENT_TYPE_TYPE,          /*EVENT_TYPE type*/
  AN_EVENT_TYPE,            /*EVENT type*/
  MPO_TYPE,                 /*MPO type*/
  MPOCLUSTER_TYPE,          /*MPOCluster type*/
} NODE_TYPE, *PNODE_TYPE;

typedef struct _NODE
{
  void  * pContent;           /*Pointer to the node content*/
  _NODE * pNextNode;    /*Point to the next node*/
```

```
  _NODE * pPrevNode;     /*Point to the previous node*/
}NODE, *PNODE;

typedef struct _LINKEDLIST
{
  PNODE  pListNode;    /*Point to the node list*/
  PNODE  pCurrent;     /*Point to the current node*/
  PNODE  pLastNode;    /*Point to the Last node*/
  int    nCount;               /*The count of the nodes*/
  NODE_TYPE nNodeType; /*The type of the data stored in the linked list*/
}LINKEDLIST, *PLINKEDLIST;

/*****************************************************************************
                function prototype list
*****************************************************************************/

PLINKEDLIST initializeLinkedList(NODE_TYPE nNodeType);
void  freeOneNode(NODE_TYPE nNodeType, PNODE pListNode);
int  deleteNodesByValuefromList(PLINKEDLIST pLinkedList, PNODE pNode, int bAll);
void  removeNodefromList(PLINKEDLIST pLinkedList, PNODE pNode);
void  destroyLinkedList(PLINKEDLIST pLinkedList);
PNODE duplicateNode(PNODE pNode, NODE_TYPE nNodeType);
PLINKEDLIST duplicateLinkedList(PLINKEDLIST ptrList);
int   nodeValueCompare(NODE_TYPE nNodeType, PNODE pNode1, PNODE pNode2);
int   addNodeToListTail(PLINKEDLIST pLinkedList, PNODE pNode);
int   addNodeToListHead(PLINKEDLIST pLinkedList, PNODE pNode);
int   addNodeToListInAscOrder(PLINKEDLIST pLinkedList, PNODE pNode);
int   equalList(PLINKEDLIST pLinkedList1, PLINKEDLIST pLinkedList2);
int   searchNodeInList(PLINKEDLIST pLinkedList, PNODE pNode1, PNODE *pNode2);
PNODE makeANodeFromValue(NODE_TYPE nNodeType, void * ptrValue);
PNODE getNodeFromIndex(PLINKEDLIST pLinkedList, int nIndex);
int combineTwoLinkedLists(PLINKEDLIST pLinkedList1, PLINKEDLIST pLinkedList2);

#endif
```

linkedList.cpp

```
/*****************************************************************************
                Macro define section
*****************************************************************************/

/*****************************************************************************
                include section
*****************************************************************************/
#include <malloc.h>
#include "linkedList.h"
#include "episode.h"

/*****************************************************************************
                Global variables
*****************************************************************************/


/*****************************************************************************
                Local function prototype list
*****************************************************************************/

/*****************************************************************************
  Description:
    This function creates and initialize a linked list.

  Input:
    nNodeType: Type for the data that will be stored in the linked list

  Returned value:
    A pointer to the allocated and initialized linked list

  History:
    02/28/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
PLINKEDLIST initializeLinkedList(NODE_TYPE nNodeType)
{
  PLINKEDLIST pLinkedList = NULL;

  pLinkedList = (PLINKEDLIST)malloc(sizeof(LINKEDLIST));

  if(pLinkedList == NULL)
    return NULL;

  /*if the linked list structure is created successfully, then assign the
    default values to the structure
  */
  if (pLinkedList)
  {
    pLinkedList->pListNode  = NULL;
    pLinkedList->pCurrent   = NULL;
    pLinkedList->pLastNode  = NULL;
    pLinkedList->nCount     = 0;
```

```
    pLinkedList->nNodeType = nNodeType;
  }
  return pLinkedList;
}

/*****************************************************************************
  Description:
    This function frees memory of a node in a linked list.
    if the linked list node is a user defined pointer and has pointers
    in it, please add code to free the dynamically allocated memory to
    avoid the memory leakage.

  Input:
    nNodeType: The type of the data that are stored in the linked list

  Returned value:
    void

  History:
    03/01/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void freeOneNode(NODE_TYPE nNodeType, PNODE pListNode)
{
    char*       ptrChar          = NULL;
    PMPOCLUSTER pStructMPOCluster = NULL;
    PMPO        pStructMPO        = NULL;

    if(pListNode == NULL)
      return;

    pListNode->pNextNode = NULL;
    pListNode->pPrevNode = NULL;

    if(pListNode->pContent == NULL)
    {
      free(pListNode);
      return;
    }

    switch(nNodeType)
    {
      case INT_TYPE:
      case UNSIGNED_INT_TYPE:
      case LONG_TYPE:
      case UNSIGNED_LONG_TYPE:
      case SHORT_TYPE:
      case UNSIGNED_SHORT_TYPE:
      case FLOAT_TYPE:
      case DOUBLE_TYPE:
      case CHAR_Type:
      case STRING_TYPE:
        free(pListNode->pContent);
        break;
```

```
        /* add code for user defined structures here. The value of the content is
           the memory address of the dynamically allcoated structure
        */

        case EVENT_TYPE_TYPE:
          free(((PEVENT_TYPE)(pListNode->pContent))->pszDescription);
          free(((PEVENT_TYPE)(pListNode->pContent))->pszEvent);
          free(pListNode->pContent);
          break;

        case AN_EVENT_TYPE:
          ((PEVENT)(pListNode->pContent))->nTime = 0;
          ((PEVENT)(pListNode->pContent))->nPosInEventList = 0;
          ((PEVENT)(pListNode->pContent))->nPosInEventTypeList = 0;
          free(pListNode->pContent);
          break;

        case MPOCLUSTER_TYPE:
          pStructMPOCluster = (PMPOCLUSTER)(pListNode->pContent);
          pStructMPOCluster->nStartPosInEventList = 0;
          destroyLinkedList(pStructMPOCluster->pEndingPosInEventList);
          pStructMPOCluster->pEndingPosInEventList = NULL;
          free(pListNode->pContent);
          break;

        case MPO_TYPE:
          pStructMPO = (PMPO)(pListNode->pContent);
          destroyLinkedList(pStructMPO->pEpisodeEventList);
          pStructMPO->pEpisodeEventList = NULL;
          destroyLinkedList(pStructMPO->pMPOClusterList);
          pStructMPO->pMPOClusterList = NULL;
          pStructMPO->nTotalOccurrence = 0;
          pStructMPO->nFLM = 0;
          free(pListNode->pContent);
          break;

        default:
          /*Warning messages here for unsupported data types*/
          writeLog(WARNING_MSG, "Data type %d not supported. File %s, line %d.",
            nNodeType, __FILE__, __LINE__);
          break;
      }

      free(pListNode);
}

/*****************************************************************************
  Description:
    This function destroys a linked list.
    if the linked list node is a user defined pointer and has pointers
    in it, please add code to free the dynamically allocated memory to
    avoid the memory leakage.
```

```
  Input:
    pLinkedList: Pointer to a linked list that needs to be destroyed

  Returned value:

  History:
    02/28/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void  destroyLinkedList(PLINKEDLIST pLinkedList)
{
   PNODE   pListNode = NULL, pNextNode = NULL;

   if(!pLinkedList)
     return;

  for (pListNode = pLinkedList->pListNode; pListNode; pListNode = pNextNode)
  {
    pNextNode = pListNode->pNextNode;

    freeOneNode(pLinkedList->nNodeType, pListNode);
  }

  pLinkedList->pCurrent = NULL;
  pLinkedList->pLastNode = NULL;

  free(pLinkedList);
}



/*****************************************************************************
  Description:
    This function copys the value of a linked list node to another one.

  Input:
    pNode1: The target node that will get new values
    pNode2: The node that has the values
    nNodeType: Type for the data that are stored in the linked list

  Returned value:
    A pointer to the newly allocated NODE structure

  History:
    02/28/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
PNODE duplicateNode(PNODE pNode, NODE_TYPE nNodeType)
{
  return makeANodeFromValue(nNodeType, pNode->pContent);
}

/*****************************************************************************
  Description:
```

```
    This function makes a duplication of one linked list.

  Input:
    ptrList: The target node that will get new values

  Returned value:
    Pointer to the new linked list

  History:
    04/08/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
PLINKEDLIST duplicateLinkedList(PLINKEDLIST ptrList)
{
  PLINKEDLIST ptrNewList = NULL;
  PNODE       pNode      = NULL;

  if(!ptrList)
  {
    //writeLog(WARNING_MSG, "The source linked list is empty. File %s,line %d.",
      // __FILE__, __LINE__);
    return ptrNewList;
  }

  ptrNewList = initializeLinkedList(ptrList->nNodeType);

  for (pNode = ptrList->pListNode; pNode; pNode = pNode->pNextNode)
    addNodeToListTail(ptrNewList, pNode);

  return ptrNewList;
}


/*****************************************************************************
  Description:
    This function compares the value of two node. This could be used in the
    sorted linked list.

  Input:
    nNodeType: The type of the data that are stored in the linked list
    node1: The first node in the comparision
    node1: The second node in the comparision

  Returned value:
    when the value of node1 is greater than node2, then return 1;
    when the value of node1 is less than node2, then return -1;
    when the value of node1 is equal to node2, then return 0;

  History:
    03/01/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int nodeValueCompare(NODE_TYPE nNodeType, PNODE pNode1, PNODE pNode2)
{
  double dNodeValue1 = 0.0, dNodeValue2 = 0.0;
```

```
int bNumeric = 0;

  /*handle the special values to avoid memory errors*/
if((pNode1 == NULL) && (pNode2 == NULL))
  return 0;

if(pNode1 == NULL)
  return 1;

if(pNode2 == NULL)
  return -1;


/*for the string data*/
if(nNodeType ==  STRING_TYPE)
{
  return strcmp((char*)pNode1->pContent, (char*)pNode2->pContent);
}

/*Add the code for other user defined data types*/
if(nNodeType == INT_TYPE)
{
  dNodeValue1 = *((int*) pNode1->pContent);
  dNodeValue2 = *((int*) pNode2->pContent);
  bNumeric = 1;
}
if(nNodeType ==  UNSIGNED_INT_TYPE)
{
  dNodeValue1 = *((unsigned int*) pNode1->pContent);
  dNodeValue2 = *((unsigned int*) pNode2->pContent);
  bNumeric = 1;

}
if(nNodeType == LONG_TYPE)
{
  dNodeValue1 = *((long*) pNode1->pContent);
  dNodeValue2 = *((long*) pNode2->pContent);
  bNumeric = 1;

}
if(nNodeType ==  UNSIGNED_LONG_TYPE)
{
  dNodeValue1 = *((unsigned long*) pNode1->pContent);
  dNodeValue2 = *((unsigned long*) pNode2->pContent);
  bNumeric = 1;

}
if(nNodeType == SHORT_TYPE)
{
  dNodeValue1 = *((short*) pNode1->pContent);
  dNodeValue2 = *((short*) pNode2->pContent);
  bNumeric = 1;
```

```
}
if(nNodeType ==  UNSIGNED_SHORT_TYPE)
{
  dNodeValue1 = *((unsigned short*) pNode1->pContent);
  dNodeValue2 = *((unsigned short*) pNode2->pContent);
  bNumeric = 1;

}
if(nNodeType == FLOAT_TYPE)
{
  dNodeValue1 = *((float*) pNode1->pContent);
  dNodeValue2 = *((float*) pNode2->pContent);
  bNumeric = 1;

}
if(nNodeType ==  DOUBLE_TYPE)
{
  dNodeValue1 = *((double*) pNode1->pContent);
  dNodeValue2 = *((double*) pNode2->pContent);
  bNumeric = 1;
}


/*User needs to convert structure to numeric value here*/
if(nNodeType == EVENT_TYPE_TYPE)
{
  /*Always return 1. Can be changed if needed*/
  return 1;
}

if(nNodeType == AN_EVENT_TYPE)
{
  dNodeValue1 = ((PEVENT)(pNode1->pContent))->nPosInEventList;
  dNodeValue2 = ((PEVENT)(pNode2->pContent))->nPosInEventList;
  bNumeric = 1;
}

if(nNodeType == MPO_TYPE)
{
  /*Always return 1. Can be changed if needed*/
  return 1;
}

if(nNodeType == MPOCLUSTER_TYPE)
{
  dNodeValue1 = ((PMPOCLUSTER)(pNode1->pContent))->nStartPosInEventList;
  dNodeValue2 = ((PMPOCLUSTER)(pNode2->pContent))->nStartPosInEventList;
  bNumeric = 1;
}

if (nNodeType == STRING_TYPE) /*For character strings*/
{
  return strcmp((char*)(pNode1->pContent), (char*)(pNode2->pContent));
```

```
  }

  if(bNumeric)
  {
    if(dNodeValue1 > dNodeValue2)
      return 1;
    else if(dNodeValue1 == dNodeValue2)
      return 0;
    else
      return -1;
  }

  writeLog(ERROR_MSG, "Data type %d not supported. File %s,line %d.",
     nNodeType, __FILE__, __LINE__);

  exit(1);
}

/*****************************************************************************
  Description:
    This function deletes the nodes in the linked list that has the same
    value of the node passed in as parameter.

  Input:
    pLinkedList: The linked list that nodes need to be removed
    pNode: The nodes in the linked list that have the same value of this one
         need to be removed.
    bAll: Only the first node with the same value will be removed if it is 0,
        otherwise all nodes with the same  value willbe removed.

 Returned value:
    Return 1 when success
    return 0 when the nodes cannot be found in the linked list;

  History:
    03/01/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int deleteNodesByValuefromList (PLINKEDLIST pLinkedList, PNODE pNode, int bAll)
{
  PNODE pListNode   = NULL;
  int   bSuccess    = 0;

  ASSERT(pLinkedList);
  ASSERT(pNode);

  for (pListNode = pLinkedList->pListNode; pListNode; pListNode = pListNode->pNextNode)
  {

    if (nodeValueCompare(pLinkedList->nNodeType, pListNode, pNode) == 0)
    {
      bSuccess = 1;

      removeNodefromList(pLinkedList, pListNode);
```

```
      /*if !bAll, we only removed the first node that has the same value*/
      if(!bAll)
        break;
    }
  }

  return bSuccess;
}

/*****************************************************************************
  Description:
    This function deletes one nodes in the linked list.

  Input:
    pLinkedList: The linked list that nodes need to be removed
    pNode: The nodes in the linked list that needs to be removed.

 Returned value:
    Return 1 when success
    return 0 when the nodes cannot be found in the linked list;

  History:
    04/07/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void  removeNodefromList(PLINKEDLIST pLinkedList, PNODE pNode)
{
  ASSERT(pLinkedList);
  ASSERT(pNode);

  /*if it node is at the head*/
  if(pLinkedList->pListNode == pNode)
  {
    pLinkedList->pListNode = pNode->pNextNode;

    if (pNode->pNextNode)
      pNode->pNextNode->pPrevNode = NULL;

    if( pLinkedList->pLastNode == pNode)
      pLinkedList->pLastNode = pNode->pPrevNode;
  }
  /*if it node is at the tail*/
  else if (pNode == pLinkedList->pLastNode)
  {
    pLinkedList->pLastNode = pNode->pPrevNode;
    pLinkedList->pLastNode->pNextNode = NULL;
  }
   /*if it node is in the middle*/
  else
  {
    pNode->pPrevNode->pNextNode = pNode->pNextNode;
    pNode->pNextNode->pPrevNode = pNode->pPrevNode;
  }
```

```
  freeOneNode(pLinkedList->nNodeType, pNode);
  pLinkedList->nCount--;

}

/*****************************************************************************
  Description:
    This function adds a node to the tail of the linked list. The function
    will relocate memory for the node again.

  Input:
    pLinkedList: The linked list that node needs to be added
    pNode: The node that needs to be added to the linked list

 Returned value:
    Return 1 when success
    return 0 when the node cannot be added to the linked list;

  History:
    03/05/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int addNodeToListTail(PLINKEDLIST pLinkedList, PNODE pNode)
{
  PNODE ptrTem = NULL;

  if(pLinkedList == NULL)
    return 0;

  ptrTem = duplicateNode(pNode, pLinkedList->nNodeType);

  if(ptrTem == NULL)
    return 0;

  ptrTem->pNextNode = NULL;

  if(pLinkedList->pListNode == NULL)
  {
    pLinkedList->pListNode = pLinkedList->pLastNode = ptrTem;
    ptrTem->pPrevNode = ptrTem->pNextNode = NULL;
  }
  else
  {
    pLinkedList->pLastNode->pNextNode = ptrTem;
    ptrTem->pPrevNode = pLinkedList->pLastNode;
    pLinkedList->pLastNode = ptrTem;
  }

  pLinkedList->nCount++;
  return 1;

}
```

```
/*****************************************************************************
  Description:
    This function adds a node to the head of the linked list. The function
    will relocate memory for the node again.

  Input:
    pLinkedList: The linked list that node needs to be added
    pNode: The node that needs to be added to the linked list

 Returned value:
    Return 1 when success
    return 0 when the node cannot be added to the linked list;

  History:
    03/05/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int addNodeToListHead(PLINKEDLIST pLinkedList, PNODE pNode)
{
  PNODE ptrTem = NULL;

  ptrTem = duplicateNode(pNode, pLinkedList->nNodeType);

  ptrTem->pNextNode = pLinkedList->pListNode;
  ptrTem->pPrevNode = NULL;
  pLinkedList->pListNode = ptrTem;
  pLinkedList->nCount++;

  return 1;
}

/*****************************************************************************
  Description:
    This function adds a node to the head of the linked list in ascending
    order. The function will relocate memory for the node again.

  Input:
    pLinkedList: The linked list that node needs to be added
    pNode: The node that needs to be added to the linked list

 Returned value:
    Return 1 when success
    return 0 when the node cannot be added to the linked list;

  History:
    03/05/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int addNodeToListInAscOrder(PLINKEDLIST pLinkedList, PNODE pNode)
{
  PNODE ptrTem     = NULL;
  PNODE ptrCurrent  = NULL;

  ptrTem = duplicateNode(pNode, pLinkedList->nNodeType);
```

```
 /*The linked list used to be empty*/
 if(pLinkedList->pListNode == NULL)
 {
    ptrTem->pNextNode = NULL;
    ptrTem->pPrevNode = NULL;
    pLinkedList->pListNode = ptrTem;
    pLinkedList->nCount = 1;
    return 1;
 }

 /*The value of the node is greater than the last node of linked list*/
 if ((pLinkedList->pListNode != NULL) &&
    (nodeValueCompare(pLinkedList->nNodeType, ptrCurrent, pNode) > 0))
 {
    ptrTem->pNextNode = NULL;
    ptrTem->pPrevNode = pLinkedList->pLastNode;
    pLinkedList->pLastNode = ptrTem;
    pLinkedList->nCount++;
    return 1;
 }

  for (ptrCurrent = pLinkedList->pListNode;
      ptrCurrent != NULL; ptrCurrent = ptrCurrent->pNextNode)
  {

     /* When the value of the current node is larger or equal than the new
        node, Then the new node will be added before the current one*/
     if (nodeValueCompare(pLinkedList->nNodeType, ptrCurrent, pNode) > -1)
       break;
  }

  if (ptrCurrent == pLinkedList->pListNode)
  {
    ptrTem->pNextNode = NULL;
    ptrTem->pPrevNode = NULL;
    pLinkedList->pListNode = ptrTem;
  }
  else
  {
    ptrTem->pNextNode = ptrCurrent;
    ptrTem->pPrevNode = ptrCurrent->pPrevNode;
    ptrCurrent->pPrevNode = ptrTem;
  }

  pLinkedList->nCount++;
  return 1;
}


/*****************************************************************************
  Description:
     This function determines if the two linked list are equal or not
```

```
  Input:
    pLinkedList1: The first linked listin the comparsion
    pLinkedList2: The second linked listin the comparsion

 Returned value:
    Return 1 when they are equal, 0 otherwise

  History:
    03/12/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/

int equalList(PLINKEDLIST pLinkedList1, PLINKEDLIST pLinkedList2)
{
  PNODE pNode1 = NULL;
  PNODE pNode2 = NULL;

  /*The node type and count must be equal*/
  if(!((pLinkedList1->nNodeType == pLinkedList2->nNodeType)  &&
    (pLinkedList1->nCount == pLinkedList2->nCount)))
  {
    return 0;
  }

  pNode1 = pLinkedList1->pListNode;
  pNode2 = pLinkedList2->pListNode;

  for (; pNode1 && pNode2; pNode1 = pNode1->pNextNode, pNode2 = pNode2->pNextNode)
  {

    if(nodeValueCompare(pLinkedList1->nNodeType, pNode1, pNode2) != 0)
      return 0;
  }

  if (pNode1 == NULL && pNode2 == NULL)
    return 1;
  else
    return 0;
}

/*****************************************************************************
  Description:
    This function searches the FIRST node that has the same value in the
    linked list

  Input:
    pLinkedList: The linked list
    pNode1:     The node is searched for
    pNode2:     The node in the linked list if found
 Returned value:
    The index of the node in the linked list if found, otherwise return -1

  History:
    03/14/2007: Created by Chuanwu (Steven) Xiong
```

```
*****************************************************************************/
int searchNodeInList(PLINKEDLIST pLinkedList, PNODE pNode1, PNODE *pNode2)
{
  PNODE ptrTemNode  = NULL;
  int nIndex        = -1;
  int bFound        = 0;

  *pNode2 = NULL;

  if (!(pLinkedList && pNode1))
  {
    writeLog(ERROR_MSG, "Invalid paramters passed in. File %s, line %d.",
      __FILE__, __LINE__);
    *pNode2 = NULL;
    return nIndex;
  }

  ptrTemNode = pLinkedList->pListNode;

  for (; ptrTemNode!= NULL; ptrTemNode = ptrTemNode->pNextNode)
  {
    nIndex++;

    if(nodeValueCompare(pLinkedList->nNodeType, ptrTemNode, pNode1) == 0)
    {
      bFound = 1;
      break;
    }
  }

  if(bFound)
  {
    *pNode2 = ptrTemNode;
    return nIndex;
  }
  else
    return -1;
}


/*****************************************************************************
  Description:
    This function makes a node from the value provided

  Input:
    nNodeType:  The data type that is stored in the node
    ptrValue:   The value of the node

 Returned value:
    A new node.

  History:
    03/15/2007: Created by Chuanwu (Steven) Xiong
```

```
**************************************************************************/
PNODE makeANodeFromValue(NODE_TYPE nNodeType, void * ptrValue)
{
  PNODE ptrNode = NULL;
  int       n           = 0;
  PLINKEDLIST ptrLinkedList1 = NULL;
  PLINKEDLIST ptrLinkedList2 = NULL;
  PEVENT     pStructEvent1  = NULL;
  PEVENT     pStructEvent2  = NULL;
  PEVENT_TYPE pStructEventType1  = NULL;
  PEVENT_TYPE pStructEventType2  = NULL;
  PMPOCLUSTER pStructMPOCluster1 = NULL;
  PMPOCLUSTER pStructMPOCluster2 = NULL;
  PMPO       pStructMPO1       = NULL;
  PMPO       pStructMPO2       = NULL;


  ptrNode = (PNODE)malloc(sizeof(NODE));

  if(!ptrNode)
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);

    return NULL;
  }

  switch(nNodeType)
  {
    case INT_TYPE:
      ptrNode->pContent = malloc(sizeof(int));
      if(!(ptrNode->pContent))
      {
        writeLog(ERROR_MSG, "Memory error. File %s, line %d",
          __FILE__, __LINE__);

        free(ptrNode);
        return NULL;
      }
      memcpy(ptrNode->pContent, ptrValue, sizeof(int));
      break;

    case UNSIGNED_INT_TYPE:
      ptrNode->pContent = malloc(sizeof(unsigned int));
      if(!(ptrNode->pContent))
      {
        writeLog(ERROR_MSG, "Memory error. File %s, line %d",
          __FILE__, __LINE__);
        free(ptrNode);
        return NULL;
      }
      memcpy(ptrNode->pContent, ptrValue, sizeof(unsigned int));
      break;
```

```
case LONG_TYPE:
  ptrNode->pContent = malloc(sizeof(long));
  if(!(ptrNode->pContent))
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);
    free(ptrNode);
    return NULL;
  }
  memcpy(ptrNode->pContent, ptrValue, sizeof(long));
  break;

case UNSIGNED_LONG_TYPE:
  ptrNode->pContent = malloc(sizeof(unsigned long));
  if(!(ptrNode->pContent))
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);
    free(ptrNode);
    return NULL;
  }
  memcpy(ptrNode->pContent, ptrValue, sizeof(unsigned long));
  break;

case SHORT_TYPE:
  ptrNode->pContent = malloc(sizeof(short));
  if(!(ptrNode->pContent))
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);
    free(ptrNode);
    return NULL;
  }
  memcpy(ptrNode->pContent, ptrValue, sizeof(short));
  break;

case UNSIGNED_SHORT_TYPE:
  ptrNode->pContent = malloc(sizeof(unsigned short));
  if(!(ptrNode->pContent))
  {
    writeLog(ERROR_MSG, "Memory error. File %s, line %d",
      __FILE__, __LINE__);
    free(ptrNode);
    return NULL;
  }
  memcpy(ptrNode->pContent, ptrValue, sizeof(unsigned short));
  break;

case FLOAT_TYPE:
  ptrNode->pContent = malloc(sizeof(float));
  if(!(ptrNode->pContent))
  {
```

```
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }
    memcpy(ptrNode->pContent, ptrValue, sizeof(float));
    break;

  case DOUBLE_TYPE:
    ptrNode->pContent = malloc(sizeof(double));
    if(!(ptrNode->pContent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }
    memcpy(ptrNode->pContent, ptrValue, sizeof(double));
    break;

  case CHAR_Type:
    ptrNode->pContent = malloc(sizeof(char));
    if(!(ptrNode->pContent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }
    memcpy(ptrNode->pContent, ptrValue, sizeof(char));
    break;

  /*character string is stored in the dynamically allocated memory*/
  case STRING_TYPE:
    n = strlen((char*)(ptrValue))+ 1;
    ptrNode->pContent = (void*) malloc(sizeof(char)*n);
    if(!(ptrNode->pContent))
    {
      writeLog(ERROR_MSG, "Memory erro. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }
    memcpy(ptrNode->pContent, ptrValue, sizeof(char)*n);
    break;

  /* add code for user defined structures here. The value of the content
    is the memory address of the dynamically allcoated structure
  */
  case AN_EVENT_TYPE:
    ptrNode->pContent = (void*) malloc(sizeof(EVENT));
    if(!(ptrNode->pContent))
    {
```

```
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }

    pStructEvent1 = (PEVENT)(ptrNode->pContent);
    pStructEvent2 = (PEVENT)(ptrValue);

    pStructEvent1->nTime = pStructEvent2->nTime;
    pStructEvent1->nPosInEventList = pStructEvent2->nPosInEventList;
    pStructEvent1->nPosInEventTypeList
      = pStructEvent1->nPosInEventTypeList;

    break;

  case EVENT_TYPE_TYPE:
    ptrNode->pContent = (void*) malloc(sizeof(EVENT_TYPE));
    if(!(ptrNode->pContent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }

    pStructEventType1 = (PEVENT_TYPE)(ptrNode->pContent);
    pStructEventType2 = (PEVENT_TYPE)(ptrValue);

    n = strlen(pStructEventType2->pszEvent) + 1;
    pStructEventType1->pszEvent = (char*)malloc(n*sizeof(char*));

    if(!(pStructEventType1->pszEvent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode->pContent);
      free(ptrNode);
      return NULL;
    }

    n = strlen(pStructEventType2->pszDescription) + 1;
    pStructEventType1->pszDescription = (char*)malloc(n*sizeof(char*));

    if(!(pStructEventType1->pszEvent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode->pContent);
      free(ptrNode);
      free(pStructEventType1->pszEvent);
      return NULL;
    }
```

```
    strcpy(pStructEventType1->pszDescription,
      pStructEventType2->pszDescription);

    strcpy(pStructEventType1->pszEvent, pStructEventType2->pszEvent);
    break;

  case MPO_TYPE:
    ptrNode->pContent = (void*) malloc(sizeof(MPO));

    if(!(ptrNode->pContent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }

    pStructMPO1 = (PMPO)(ptrNode->pContent);
    pStructMPO2 = (PMPO)(ptrValue);

    /*Make the episode event list*/
    ptrLinkedList2 = pStructMPO2->pEpisodeEventList;
    ptrLinkedList1 = duplicateLinkedList(ptrLinkedList2);
    pStructMPO1->pEpisodeEventList = ptrLinkedList1;

    /*Make the MPO cluster list*/
    ptrLinkedList2 = pStructMPO2->pMPOClusterList;
    ptrLinkedList1 = duplicateLinkedList(ptrLinkedList2);
    pStructMPO1->pMPOClusterList = ptrLinkedList1;

    pStructMPO1->nTotalOccurrence = pStructMPO2->nTotalOccurrence;
    pStructMPO1->nFLM = pStructMPO2->nFLM;
    break;

  case MPOCLUSTER_TYPE:
    ptrNode->pContent = (void*) malloc(sizeof(MPOCLUSTER));
    if(!(ptrNode->pContent))
    {
      writeLog(ERROR_MSG, "Memory error. File %s, line %d",
        __FILE__, __LINE__);
      free(ptrNode);
      return NULL;
    }

    pStructMPOCluster1 = (PMPOCLUSTER)(ptrNode->pContent);
    pStructMPOCluster2 = (PMPOCLUSTER)(ptrValue);

    pStructMPOCluster1->nStartPosInEventList
      = pStructMPOCluster2->nStartPosInEventList;

    ptrLinkedList2 = pStructMPOCluster2->pEndingPosInEventList;
```

```
        ptrLinkedList1 = duplicateLinkedList(ptrLinkedList2);
        pStructMPOCluster1->pEndingPosInEventList = ptrLinkedList1;
        break;

      default:
        break;
    }

    ptrNode->pNextNode = NULL;
    ptrNode->pPrevNode = NULL;

    return ptrNode;
}


/*****************************************************************************
    Description:
      This function retrives the node in a linked list by the index. Make sure
      this function return a valid pointer to a node. Otherwise, it is an
      error.

    Input:
      pLinkedList: The linked list
      nIndex:      The index number

    Returned value:
      a valid pointer to the node in the linked list

    History:
      04/10/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
PNODE getNodeFromIndex(PLINKEDLIST pLinkedList, int nIndex)
{
    PNODE ptrNode  = NULL;
    int   nCount   = 0;

    /*Make sure the pointers are not NULL*/
    if(!pLinkedList)
    {
      writeLog(ERROR_MSG, "The linked list is empty. File %s, line %d",
        __FILE__, __LINE__);
      return NULL;
    }

    /*Make sure the index nuber is valid*/
    if(nIndex > pLinkedList->nCount)
    {
      writeLog(ERROR_MSG, "The index number %d is greater than the size "
        "of the linked list %d. File %s, line %d",
        nIndex, pLinkedList->nCount, __FILE__, __LINE__);
      return NULL;
    }
```

```
  ptrNode = pLinkedList->pListNode;

  for (; ptrNode; ptrNode= ptrNode->pNextNode)
  {
      if(nIndex == nCount)
        break;

      nCount++;
  }

  if(!ptrNode)
  {
    writeLog(ERROR_MSG, "The node is not valid. File %s, line %d",
      __FILE__, __LINE__);
    return NULL;
  }

  return ptrNode;
}

/*****************************************************************************
  Description:
    This function add the all the nodes in the second linked list to the
    first one. Make sure that the two linked list are for the same data
    before calling this founction.

  Input:
    pLinkedList1: The first linked list
    pLinkedList2: The second LinkedList

  Returned value:
    1 when successful, 0 otherwise

  History:
    04/11/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
int combineTwoLinkedLists(PLINKEDLIST pLinkedList1, PLINKEDLIST pLinkedList2)
{
  PNODE pNode = NULL;

  /*If the second linked list is NULL or empty, do nothing*/
  if (pLinkedList2 == NULL || pLinkedList2->nCount == 0)
    return 1;

  /*Make sure the first linked list is not NULL*/
  if (pLinkedList1 == NULL)
  {

    writeLog(ERROR_MSG, "The pointer is invalid. File %s, line %d",
      __FILE__, __LINE__);
    exit(1);
  }
```

```
  if(!(pLinkedList1->nNodeType == pLinkedList2->nNodeType))
  {


    writeLog(ERROR_MSG, "The data types in the two linked list d not match. "
      "File %s, line %d", __FILE__, __LINE__);
    exit(1);
  }

  pNode = pLinkedList2->pListNode;
  for (; pNode; pNode = pNode->pNextNode)
  {
    if (addNodeToListTail(pLinkedList1, pNode) == 0)
      return 0;
  }

  return 1;
}
```

log.h

```
#ifndef _LOG_H
#define _LOG_H

/*****************************************************************************
                Macro define section
*****************************************************************************/
#define LOG_FILE "log"

#ifdef DEBUG
  #define ASSERT(f)  if (!f) writeFatalDebugMsgAndExit( __FILE__, __LINE__ )
#else
  #define ASSERT(f)
#endif

#ifdef WIN32
  #define getpid _getpid
#endif

/*****************************************************************************
                include section
*****************************************************************************/
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>

#ifdef WIN32
  #include <process.h>
#else
  #include <unistd.h>
#endif

/*****************************************************************************
                Struction definition section
*****************************************************************************/
typedef enum
{
        ERROR_MSG,              /*Error message*/
  WARNING_MSG,              /*Warning message*/
  INFO_MSG,                 /*information message*/
  DEBUG_MSG,                /*Debugging message*/
  NOLEAD_MSG                /*Don't put any leading words before the message*/
} LOG_TYPE, *PLOG_TYPE;


typedef enum
{
        RETURN_ERROR,               /*Error*/
  RETURN_SUCCESS,       /*Sucess*/
```

```
  RETURN_WARNING         /*There is a warning*/
} RETURN_TYPE, *PRETURN_TYPE;


/*****************************************************************************
                function prototype list
*****************************************************************************/
void writeLog(LOG_TYPE nLogType, char *szFormat, ...);
void writeFatalDebugMsgAndExit(char *szFileName, char *szLineNumber);

#endif
```

log.cpp

```
/*****************************************************************************
                Macro define section
*****************************************************************************/

/*****************************************************************************
                include section
*****************************************************************************/
#include "log.h"

/*****************************************************************************
                Global variables
*****************************************************************************/


/*****************************************************************************
                Local function prototype list
*****************************************************************************/


/*****************************************************************************
  Description:
    This function writes the fatal runtime error messages for debugging and
    exit the execution of the application. The message can be printed only
    when the DEBUG is turned on.

  Input:
    szFileName: The file name in which the fatal error happens
    szLineNumber: The line number where the fatal error occurs

  Returned value:
    void

  History:
    03/06/2007: Created by Chuanwu (Steven) Xiong
*****************************************************************************/
void writeFatalDebugMsgAndExit(char *szFileName, int nLineNumber)
{
  writeLog(DEBUG_MSG, "Fatal error occur in file %s at line %d", szFileName, nLineNumber);
  exit(1);
}


/*****************************************************************************
  Description:
    This function write messge to the log file

  Input:
    szFilePath: the file path
    nLogType: The message type of the log, such as the error, warning, infor
    szFormat: The format in which the message will be write
    ...: The variable argument list which contains the message to be written
```

```
  Returned value:
    void

  History:
    03/06/2007: Created by Chuanwu (Steven) Xiong
*******************************************************************************/
void writeLog(LOG_TYPE nLogType, char *szFormat, ...)
{
  char     szBuffer[1000]  = {0};
  char     szTime[25]      = {0};
  char     szMsg[256]      = {0};
  va_list  szVarList       = NULL;
  char     szFileName[256]= {0};
  FILE     *ptrFile        = NULL;
  int      nResult         = 0;
  struct   tm *pStructTime = NULL;
  long     ltime           = 0;

#ifdef WIN32
  struct _stat buf;
#else
  struct stat buf;
#endif

  if(szFormat == NULL)
    return;

#ifdef WIN32
  sprintf(szFileName, "log\\%s%d.log", LOG_FILE, getpid());
  nResult = _stat( "log", &buf );
#else
  sprintf(szFileName, "log/%s%d.log", LOG_FILE, getpid());
  nResult = stat("log", &buf);
#endif

  if(nResult != 0)
  {
    nResult = system("mkdir log");

   /*If cannot create the log folder, exit*/
    if(nResult != 0)
      return;
  }

  ptrFile = fopen(szFileName, "a");

  if(!ptrFile)
    return;

  va_start(szVarList, szFormat);
  vsprintf(szBuffer, szFormat, szVarList);
  va_end( szVarList);
```

```
  time( &ltime );                  /*Get the time*/
  pStructTime = localtime( &ltime );   /*Convert it to the tm structure*/

  if( pStructTime->tm_hour > 12 )      /* Convert from 24-hour */
    pStructTime->tm_hour -= 12;       /*   to 12-hour clock.  */
  if( pStructTime->tm_hour == 0 )      /*Set hour to 12 if midnight. */
    pStructTime->tm_hour = 12;

  sprintf(szTime, "%02d:%02d:%02d", pStructTime->tm_hour,
    pStructTime->tm_min, pStructTime->tm_sec);

  if( pStructTime->tm_hour > 12 )      /* Set up extension. */
    strcat(szTime, " PM" );
  else
    strcat(szTime, " AM" );

  switch(nLogType)
  {
    case ERROR_MSG:
      sprintf(szMsg, "%-18s%-10s%s", szTime, "Error:", szBuffer);
      break;

    case WARNING_MSG:
      sprintf(szMsg, "%-18s%-10s%s", szTime, "Warning:", szBuffer);
      break;

    case INFO_MSG:
      sprintf(szMsg, "%-18s%-10s%s", szTime, "Info:", szBuffer);
      break;

    case DEBUG_MSG:
      sprintf(szMsg, "%-18s%-10s%s", szTime, "Debug:", szBuffer);
      break;

    case NOLEAD_MSG:
      sprintf(szMsg, " %s",  szBuffer);

    default:
      sprintf(szMsg, "%-18s%-10s%s", szTime, " ", szBuffer);
      break;
  }

  fprintf(ptrFile, "%s\n", szMsg);

  fclose(ptrFile);
}
```

getopt.h

```
#ifndef _GETOPT_H
#define _GETOPT_H

/*For microsoft platform only. UNIX has it in the library*/
#ifdef WIN32

/*****************************************************************************
                Macro define section
*****************************************************************************/
#if M_I8086 || M_I286 || MSDOS    /* test Microsoft C definitions */
  #define           SWITCH          '/'         /*  only used for DOS */
#else
  #define           SWITCH          '-'         /* -: always recognized */
#endif

/*****************************************************************************
                include section
*****************************************************************************/
#include            <string.h>

/*****************************************************************************
                Global variable
*****************************************************************************/

/*****************************************************************************
                function prototype list
*****************************************************************************/
int getopt( int argc, char **argv, char *opts );

#endif
#endif
```

getopt.cpp

```
/*****************************************************************************
*                                                                            *
* getopt.c:       Derived from AT&T public domain source of getopt(3),       *
*                 modified for use with MS C 6.0 on MS DOS systems. For      *
*                 unknown reasons the variable optopt is exported here.      *
*                                                                            *
*     Note that each option may occur more than once in the command          *
*     line, this may require special action like occurence counting.         *
*     Each option is indicated by a single character in opts string          *
*     followed by : if an option argument is required. So for "abo:"         *
*     the following combinations are possible:                               *
*          -a -b -o value   sets a, b, and argument value for o              *
*          -ab -o value     equivalent                                        *
*          -ab -ovalue      equivalent, but not recommended                   *
*          -abovalue        equivalent, but not recommended                   *
*          -a -- -b         sets only a, optind advanced to -b                *
*          -a - -b          sets only a, optind stays at single -             *
*          -A               error message for A, returned as ?                *
*          -o               error message if no more arguments                *
*                                                                            *
* example code:                                                              *
* ...                                                                        *
*   extern int getopt( int, char **, char * );                               *
*   ...                                                                      *
*   int main( int argc, char *argv[] )                                       *
*   {                                                                        *
     *  extern int   opterr;                                                 *
*       extern int   optind;                                                 *
*       extern char *optarg;                                                 *
*       int c,     aset = 0,  bset = 0;                                      *
*       char *oarg = NULL;                                                   *
*                                                                            *
 *    while (( c == getopt( argc, argv, "abo:" ) != -1 )                     *
*       switch ( c )                                                         *
*       {          case 'a':                                                 *
*                        ++aset;  continue;                                  *
*                  case 'b':                                                 *
*                        ++bset;  continue;                                  *
*                  case 'o':                                                 *
*                        oarg = optarg;     continue;                       *
*                  default:                                                  *
*                       ...              return 1;                          *
*/           /* case '?': NOT EXPLICITLY NEEDED WITH DEFAULT                 *
*                  case ':': WILL NEVER HAPPEN, ':' NOT ALLOWED              *
*                  case '-': WILL NEVER HAPPEN, '-' NOT ALLOWED              *
*       }                                                                    *
     *  ...                                                                  *
*       }
 *
 *                                                                                *
*****************************************************************************/
```

```
/*For microsoft platform only. UNIX has it in the library*/
#ifdef WIN32


/*****************************************************************************
                Macro define section
*****************************************************************************/

/*****************************************************************************
                include section
*****************************************************************************/
#include <math.h>
#include "getopt.h"
#include "log.h"

/*****************************************************************************
                Global variables
*****************************************************************************/
char* optarg;               /* option argument if : in opts      */
int   optopt;               /* last option (export dubious)      */

/*Local global variable*/
static int  sp = 1;         /* offset within option word         */
static int  optind = 1;     /* next argv index                   */
static int  opterr = 1;     /* show error message if not 0*/

/*****************************************************************************
                Local function prototype list
*****************************************************************************/


/*****************************************************************************
  Description:
    This function write messge to the log file

  Input:
    nLogType: The message type of the log, such as the error, warning, infor
    szFormat: The format in which the message will be write
    ...: The variable argument list which contains the message to be written

  Returned value:
    void

  History:
    03/06/2007: modified by Chuanwu (Steven) Xiong
*****************************************************************************/
static int badopt( char *name, char *text )
{
  /* show error message if not 0    */
  if ( opterr )
    writeLog( ERROR_MSG, "%s: %s -- %c. File %s, line %d",  name, text,
    optopt, __FILE__, __LINE__);
```

```
  /* ?: result for invalid option */
  return (int) '?';
}


/*****************************************************************************
  Description:
    This function write messge to the log file

  Input:
    nLogType: The message type of the log, such as the error, warning, infor
    szFormat: The format in which the message will be write
    ...: The variable argument list which contains the message to be written

  Returned value:
    void

  History:

    03/06/2007: modified by Chuanwu (Steven) Xiong
*****************************************************************************/
int getopt(int argc, char **argv, char *opts )
{
        char *cp = NULL, ch = 0;

  if(sp == 1 )
        {
          if ( argc <= optind || argv[optind][1] == '\0' )
              return EOF;       /* no more words or single '-'  */

          if (( ch = argv[optind][0] ) != '-' && ch != SWITCH )
     return EOF;       /* options must start with '-'  */

          if ( ! strcmp( argv[optind], "--" ))
          {
              ++optind;                         /* to next word */
              return EOF;                       /* -- marks end */
            }
         }
         optopt = (int)(ch = argv[optind][sp]);         /* flag option      */

         if (ch == ':' || (cp = strchr(opts, ch)) == NULL)
         {
             if (argv[optind][++sp] == '\0')
          ++optind;

         sp = 1;         /* to next word */

          return badopt(argv[0], "illegal option" );
          }

          if (*++cp == ':')             /* ':' option requires argument */
     {
```

```
            optarg = &argv[optind][sp + 1];     /* if same word */
             ++optind;
             sp = 1;                            /* to next word */

             if (*optarg == '\0')                /* in next word */
             {
                    if (argc <= optind)                 /* no more word */
               return badopt(argv[0],   "option requires an argument");

               optarg = argv[optind++];                 /* to next word */
             }
         }
         else  /* flag option without argument */
         {
            optarg = NULL;

            if (argv[optind][++sp] == '\0')
               optind++;

            sp = 1;          /* to next word */
         }

         return optopt;
}

#endif /*For windows platform only*/
```

VITA

Chuanwu Xiong

Candidate for the Degree of

Master of Science

Thesis: A STUDY OF FIRST LOCAL MAXIMUM OF CONFIDENCE IN MINING SEQUENTIAL PATTERNS

Major Field: Computer Science

Biographical:

Education: Received Bachelor of Science degree in Geology from Nanjing University, P. R. China 1990; received Master of Science in Geology at Nanjing University in 1993; completed the requirements of a Master of Science in Computer Science, Oklahoma State University, Stillwater, Oklahoma in July 2007.

Experience: Geologist at Research Institute of Petroleum Exploration & Development, China National Petroleum Company, August 1993 to January, 1998; programmer and Analyst, JD. Edwards Company, March 2001 to July 2003; programmer and Analyst, PeopleSoft Company, July 2003 to January 2005.

Name: Chuanwu Xiong Date of Degree: July, 2007

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A STUDY OF FIRST LOCAL MAXIMUM OF CONFIDENCE IN MINING SEQUENTIAL PATTERNS

Pages in Study: 106 Candidate for the Degree of Master of Science
Major Field: Computer Science

Scope and Method of Study: Constraint-based data mining in sequential database

Findings and Conclusions: Sequential data mining is increasingly important in many domains. WinMiner is a constraint-based algorithm to retrieve frequent episodes and association rules of high confidence and to search first local maximum (FLM) - rules. An algorithm for mining FLM rules from sequential dataset is implemented and is applied to several datasets of different origins. The experiments show that FLM rules are rare in randomly generated dataset and loosening the mining constraints leads to the increase of numbers of FLM rules. Correlations or dependencies among the constituent events introduced into the randomly generated dataset can dramatically increase numbers of FLM rules.

ADVISER'S APPROVAL: Dr. H. K. Dai