DEVELOPMENT OF A CONSTRAINT BASED SEQUENTIAL PATTERN MINING TOOL

Thesis submitted to the
Institute for Graduate Studies in the Social Sciences
in partial fulfillment of the requirements for the degree of

Master of Arts
in
Management Information Systems

by
Gülşah Yılmaz

Boğaziçi University
2010

Thesis Abstract

Gülşah Yılmaz, "Development of a Constraint Based Sequential Pattern Mining Tool"

Sequential pattern mining is an important data mining task with wide applications. In traditional sequential pattern mining techniques, the main purpose is to discover sequences that are frequent. The information acquired from sequential pattern mining can be used in marketing, medical records, sales analysis. The main disadvantages of the traditional algorithms such as Generalized Sequential Patterns (GSP) are the lack of focus on user expectations and the high number of discovered patterns. Constraint based data mining technique is an efficient solution as it limits the patterns within a set of conditions based on the requirements of the user. Despite the importance of constraint based sequential pattern mining, a constraint based data mining tool that handles many of the user-specified constraints simultaneously does not exist in the literature.

The aim of this study is developing a constraint based sequential pattern mining algorithm and a constraint based sequential pattern mining tool CBSPM (Constraint Based Sequential Pattern Mining) which handles a set of user-specified constraints simultaneously. A data generator facility is also developed for generating artificial sequential datasets. The verification test is conducted and the sensitivity of execution times on various parameters are examined. Performance tests shoved that as the number of constraints increases performace of the tool increases.

Tez Özeti

Gülşah Yılmaz, "Kısıt Tabanlı Veri Madenciliği Aracı Geliştirme"

Ardışık zamanlı örüntü keşfi, geniş uygulamala alanları ile veri madenciliğinin önemli bir parçasıdır. Geleneksel ardışık zamanlı örüntü keşfi algoritmalarının ana amacı sıklıkla gerçekleşen örüntüleri belirlemektir. Ardışık zamanlı örüntü keşfi sonucunda edinilen bilgiler pazarlama alanında, tıbbi kayıtlarda ve satış analizi gibi alanlarda kullanılabilir. Genelleştirilmiş Ardışık Örüntüler (GSP) gibi bazı geleneksel algoritmaların kullanıcıların birtakım beklentilerini karşılayamama ve çok sayıda örüntü keşfetme gibi bazı sakıncaları bulunmaktadır. Kısıt tabanlı veri madenciliği teknikleri, sonuçları kullanıcının ihtiyaçlarına ve beklentilerine göre sınırladığı için bu soruna etkin bir çözüm oluşturmaktadır. Kısıt tabanlı veri madenciliği son derece önemli olmasına rağmen, kullanıcı tarafından belirlenmiş kısıtları eş zamanlı olarak ele alan bir veri madenciliği aracı literatürde bulunmamaktadır.

Bu çalışmanın amacı, kullanıcı tarafından belirlenmiş kısıtları eş zamanlı olarak ele alan ve bu kısıtlara göre sonuç üreten Kısıt Tabanlı Ardışık Örüntü Keşfi algoritması ve bir Kısıt Tabanlı Ardışık Örüntü Keşfi aracı geliştirmektir. Aynı zamanda yapay sıralı veri setleri üretmek amacıyla Veri Üretme yapısı da geliştirilmiştir. Algoritma ve aracın doğrulama testleri yapılmış ve çalışma zamanının çeşitli parametrelere göre duyarlılığı test edilmiştir. Doğrulama testi sonucunda Kısıt Tabanlı Ardışık Örüntü Keşfi aracının işlevselliği kanıtlanmış ve performans testleri sonucunda kısıtlar olmadan Kısıt Tabanlı Ardışık Örüntü Keşfi aracının performansının düştüğü kısıtlar eklendikçe performansın giderek arttığı görülmüştür.

ACKNOWLEDGEMENTS

First of all I would like to thank to my supervisors, Assist. Prof. Bertan Badur and Assist. Prof. Sona Mardikyan, for their extreme support and motivation. I am also thankful to my jury members, Assoc. Prof. Osman N. Darcan, Assoc. Prof. Eyüp Çetin and Assit. Prof. Eylem Deniz.

I wish to specially thank İsmail Benk, for his support not only in this long process but also during all my life, for encouraging me throughout my thesis with patience and for sharing his valuable comments and experiences about the design and building the application.

I am deeply grateful to all the staff members of MIS department especially Figen Bacıoğlu, Sevilay Barış and Gökçen Çırakoğlu to my instructors and colleagues, especially to my dear friends Yildiz Akkaya, Can Aytekin, Osman Yücel and Ümit Topaçan for their motivation, and understanding.

I would also like to thank my managers Mustafa Elbir, Fatih Çakmak and my colleague Eray Uluhan in MSU for their precious support and comments, without their sympathy I could not finish my study.

I owe my special thanks to my parents especially to my mother Ayşe Yılmaz, for their encouragement, endless belief and understanding.

Finally, I would like to express my special thanks to TUBITAK (The Scientific and Technical Research Council of Turkey) for supporting me during my graduate study.

CONTENTS

FIGURES

CHAPTER 1

INTRODUCTION

Data mining which is recognized as the stage of the process of knowledge discovery in databases allows discovery of previously unknown but potentially useful information from the data. Sequential Pattern mining is the process of applying data mining techniques to a sequential database for the purposes of discovering the correlation relationships that exist among an ordered list of events.

In the literature, the traditional sequential pattern mining algorithms are often examined by the minimum support constraint. However, very often users want to restrict the set of patterns to be discovered according to their expectations. Constraint based data mining technique is an efficient solution as it restricts the patterns within a set of conditions based on the requirements of the user. Constraints can be studied from different points of view. From the application perspective there are seven types of user-specified constraints: *Item Constraint, Length Constraint, Super pattern Constraint, Aggregate Constraint, Duration Constraint and Gap Constraint.* By using *Item Constraint,* user may specify the item that should or should not be included in the pattern. *Length constraint* helps user to indicate the pattern length, which can be the number of times the items appear. *Super pattern Constraint* is a more general version of *Item Constraint*. It is used if the user wants to find patterns that contain a particular set of patterns. *Aggregate constraint* can be applied on a set of items in the given pattern. The aggregation function can be sum, average, maximum value, etc. *Duration Constraint* and *Gap Constraint* are the time constraints. Both of them are applicable for sequence databases where transactions have time stamps for every sequence. Although

constraint based sequential pattern mining is very important, as a result of the literature survey, there is no such a data mining tool that handles many of these user-specified constraints simutaneously.

In this study a Sequential Pattern Mining algorithm and a Sequential Pattern Mining tool that named as CBSPM-Constraint Based Sequential Pattern Mining are developed by using Java programming language, for discovering sequential patterns which enables the user to focus on the constraints. The tool has a graphical user interface (GUI) to provide user friendly environment and a data generator facility to generate artificial datasets. Subsequently the verification test is conducted and the sensitivity of execution times on various parameters are examined.

The remainder of this thesis is organized as follows. In Chapter 2 the literature review about Sequential Pattern Mining and Sequential Pattern Mining software are given. Chapter 3 states the definition of the Sequential Pattern Mining, Generalized Sequential Pattern Mining algorithm, constraints and constraint types. In Chapter 4, the definition of the problem, how CBSPM was developed and the GUI and a data generator facility are introduced in Chapter 4. In Chapter 5, performance and verification tests and the outputs of these tests are presented in detail. Finally, in Chapter 6, the conclusions drawn from the study are expressed and suggestions for the further research are given.

CHAPTER 2

LITERATURE REVIEW OF SEQUENTIAL PATTERN MINING

This chapter introduces literature review about Data mining, Sequential Pattern Mining and Sequential Pattern Mining softwares.

Data Mining

Data mining is the process of automatically discovering useful information in large databases (Han ve Kamber 2006). There are many data mining techniques which are utilized to extract specific information or patterns available from databases and other kinds of repositories.

Data mining can be classified into two main categories descriptive and predictive (Han ve Kamber 2006). Descriptive mining is to summarize general properties of data in databases, while predictive mining is to perform inference on data.

Two predictive data mining tasks can be performed types of: classification and prediction. Classification is the process of finding group membership for data instances. Decision trees, neural networks, and logistic regression are examples for classification techniques. Whilst classification predicts discrete (categorical) labels, prediction models continuous-valued functions (Han ve Kamber 2006). Regression analysis is the major model for prediction.

Association analysis, cluster analysis, outlier analysis and evolution analysis are groups of different descriptive data mining tasks (Han ve Kamber 2006).

Patterns that occur frequently and describe strongly associated features in data are called frequent patterns. Association analysis discovers the patterns of associations

and correlations represented in the form of implication rules in sequences or frequent item sets.

Sequential Pattern Mining is an extension of Association Rule Mining. Sequential pattern mining finds the relationships between occurrences of sequential events, if there exists any specific order of the occurrences. In association rule mining, the mining results are about which items are brought together frequently, those items must come from the same transaction. While the result of sequential pattern mining is about which items are brought in a certain order by the same customer, those items come from different transactions.

Sequential Pattern Mining

Sequential pattern mining aims at looking for links between events happening sequentially to see if there is any order in which they occur. The various research articles used in this thesis proceed towards the common goal of understand sequential pattern mining, study its generalized form and user-specified constraints.

Sequential Pattern Mining was introduced by Agrawal and Srikant (1995). The problem was originally described for market basket analysis for customer transaction database. Sequential pattern mining is based on the Apriori principle proposed in association rule mining (Agrawal & Srikant 1994), this principle states that any sub-pattern of a frequent pattern must be frequent. Based on this principle, a series of Apriori-like algorithms have been proposed. The research article by Srikant and Agrawal (1996) presents the Generalized Sequential Pattern (GSP) algorithm as a faster alternative to the AprioriAll algorithm which was presented by (Agrawal & Srikant 1994), In the GSP the problem was generalized by introducing the idea of time

constraint, sliding time window and taxonomy. The research article by Pei and Han, (2001), presents an in-depth study of the constraints involved in sequential pattern mining and develops a new framework for constraint based mining. The seven different kinds of constraints that are in use currently are described as well. For effectiveness and efficiency considerations, constraints are essential in many data mining applications. In the context of constraint-based sequential pattern mining, Srikant and Agrawal (1996), generalize the scope of sequential pattern mining to include time constraints, sliding time windows, and user-defined taxonomy. According to Pei et al. (2002) sequential pattern mining algorithms, in general, can be categorized into three classes: (1) Apriori-based, horizontal formatting method, with GSP as its representative; (2) Apriori-based, vertical formatting method, such as SPADE (Sequential Pattern Discovery using Equivalence classes) which was proposed by Zaki (2001) and (3) projection-based pattern growth method, such as PrefixSpan which was proposed by Pei and Han (2001), mines the complete set of patterns but greatly reduces the efforts of candidate subsequence generation.

Mining frequent episodes in a sequence of events studied by Mannila et al. (1997) can also be viewed as a constrained mining problem, since episodes are essentially constraints on events in the form of acyclic graphs. Garofalakis et al. (2002) proposes regular expressions as constraints for sequential pattern mining and develop a family of SPIRIT algorithms, while members in the family achieve various degrees of constraint enforcement. They use relaxed constraints (e.g. anti-monotonicity, monotonicity) to filter out some irrelevant patterns/candidates for user in their early stage. Ng et al. (1999) have explored the antimonotonic and succinct constraints and demonstrated how they can be used to prune the search space.

Morzy et al. (2002) , classifies the constraint processing techniques as: (1) Post processing (2) Candidate Filtering and (3) Dataset Filtering and also demonstrates how to reduce the size of the dataset while operating within the same data mining task and an extended GSP algorithm applying dataset filtering and performance gain evaluation. According to Antunes and Oliveira, (2003) the classic Generalized Sequential Patterns (GSP) algorithm returns all frequent sequences present in a database. However, usually a few ones are interesting from a user's point of view. Thus, post-processing tasks are required in order to discard uninteresting sequences and shows a study on how the PrefixSpan algorithm can be generalized so as to work with gap constraints to create generated databases with post-processing techniques.

Chen and Hu (2006), introduces a study on the concepts of recency and compactness in addition to frequency and presents a new pattern known as CFR which is a combination of all three. Chang et al. (2006), launch an algorithm known as FSPAN or fast sequential pattern mining algorithm which can effectively uses the depth first search approach while mining patterns onto large datasets. Cooley et al. (1997) explains how web based mining work and introduces the system known as webminer.

Fiot et al. (2007), presents a way to conduct generalized sequential pattern mining using a graphical approach obtained from broadened temporal constraints. Gomez and Vaisman (2009), offers a powerful language known as RE-SPaM which avoids the drawbacks of RE (Regular Expressions). This language utilizes constraints associated with the attributes of the sequences of the entities being mined and lays focus on trajectory databases.

WEKA (Waikato Environment for Knowledge Analysis) is a well known suite of machine learning software developed by using Java, at the University of Waikato, New Zealand (Witten et al, 2009). WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. In addition there is a GSP tool in WEKA.

Algorithm GeneralizedSequentialPatterns in WEKA is described as a GSP algorithm for discovering sequential patterns in a sequential dataset. According to the description, this algorithm is based on the paper: Srikant and Agrawal (1996), Mining Sequential Patterns: Generalizations and Performance Improvements. In that paper, sequences are lists of item sets, where each sequence can contain an arbitrary number of item sets and each item set can contain an arbitrary number of items. Moreover, items can correspond to item taxonomy. Although there is a GSP algorithm in WEKA, this tool does not handle any constraints including time constraints.

Rapid Miner is another data mining tool which handles the GSP algorithm. It is an open-source data mining system. This operator searches sequential patterns in a set of transactions. Each transaction must be encoded as a single example and must contain one attribute for the time and for the customer. This pair of attribute is used for generate one sequence per customer containing each single transaction ordered by the time of each transaction. The algorithm then searches sequential patterns (Mierswa et al, 2006).

There is another Sequential Pattern Mining tool SPIRIT (termed as Sequential Pattern Mining with Regular Expression Constraints) which was proposed by Garofalakis et al. (1999). This tool only satisfies user-specified Regular Expression

constraints. Regular Expression Constraint is one of the constraint types which was defined by Pei et al. (2002).

CHAPTER 3

SEQUENTIAL PATTERN MINING

In this chapter, basic concepts of sequential pattern mining, generalized sequential pattern mining algorithm, constraints and constraint types are summarized.

Sequential Pattern Mining

The problem of sequential pattern mining is one of the data mining approaches that have deserved specific attention. It was first introduced by Agrawal and Srikant (1995) based on their study of customer purchase sequences. It is used in many other applications such as web-log mining, bio-sequence analysis, medical treatment sequence analysis.

Sequential pattern mining attempts to find inter-session patterns such as the existence of a set of items followed by another item in a time-ordered set of sessions (Agrawal & Srikant, 1995).

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of *items*. A sequence $s = e_1 e_2 ... e_j$ is an ordered list of elements, where each e_i $(1 \leq i \leq j)$ is an element or itemset of the sequence; an element is denoted as $e=(i_{k1} i_{k2} \ldots i_{kn})$, where i_{kt} $(1 \leq t \leq n)$ is an item. For the sake of brevity, the brackets are omitted if an element has only one item, that is, element (x) is written as x. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. Length of a sequence, s, is given by the number of elements of the sequence. Sequence which contains k-items is a k-sequence. A sequence $\alpha = e_1 e_2 \ldots e_n$ is called a subsequence of another sequence $\beta = e_1 e_2 ... e_m$ and β is a super sequence of α $(n \leq m)$, denoted as $\alpha \sqsubseteq \beta$, if there exist integers

$1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $e_1 \subseteq e_{j1}, e_2 \subseteq e_{j2}, \ldots, e_n \subseteq e_{jn}$.

A sequence database S is a set of rows of the form (sid, s), w0here *sid* is a sequence id and s is a sequence. A row (sid, s) is said to contain a sequence α, if α is a subsequence of s.

The support of a sequence α in a sequence database S is the number of rows in the database containing α, that is (Agrawal & Srikant, 1994),

$$support_s(\alpha) = |\ \{((sid, s)\ |\ (sid, s) \in S)\ \}\ |$$

Given a user-specified positive integer *min_support* as the support threshold, a sequence α is called a frequent sequential pattern in sequence database S if,

$$support_s(\alpha) \geq min\_support.$$

Given a sequence database and a minimum support threshold, the sequential pattern mining problem is to find the complete set of frequent sequential patterns in the database. Sequential pattern mining is based on the Apriori principle proposed in association rule mining (Agrawal & Srikant, 1994), this principle states that any sub-pattern of a frequent pattern must be frequent. Apriori principle is an effective way to eliminate candidate itemsets without counting their support values.

Generalized Sequential Pattern Mining Algorithm

A classical sequential pattern mining algorithm, GSP mines sequential patterns by adopting a candidate subsequence generation-and-test approach based on the Apriori principle.

Input: S= a sequence database

min-sup = a support threshold

Output: frequent sequences s

L_1 = the set of frequent 1-sequence

k=2, do while $L_{(k-1)}$! = Null;

C_2=combine($L_{(k-1)}$);

for all candidates c ∈ C_k (set of candidate k-sequences); // C ⊆ C_k

for all input sequences s in the database S do

if s contains c

c.count ++

L_k = {c ∈ C_k | c.count ≥ mincont} // prune phase

end if;

end;

for all frequent sequences l ∈ L_k

generate C_{k+1} do

while L_k ! = null; //candidate generation phase

end;

end;

Set of all frequent sequences is the union of all $L_{k=}$L // // L ⊆ L_k

Fig. 1 GSP Algorithm

The generalized sequential pattern mining algorithm which is shown in Figure 1 passes through the data numerous times (Srikant & Agrawal, 1996). Given the database S and the minimum support threshold *min_support*, GSP first scans S, then identifies the support value of the items and finds the set of frequent items, that is, frequent 1-length subsequences. The next stage starts with the frequent sequences (L) obtained in the former stage (called seed set where each member of L_1 represents a 1-element sequential pattern). These create a new set of frequent sequences known as candidate sequences (C). Each of these have at one additional item, more than the seed sequence. C_2 is generated from L_1.

Then, the sequence database is scanned again, and the supports of sequences in C_2 are counted. Those sequences in C_2 that are greater than the minimum support threshold are the 2-length sequential patterns. By using the 2-length sequential patterns, C_3 the set of 3-length candidates can be generated. The set of candidates is generated by a self-join of the sequential patterns found in the former pass. In the k^{th} pass, a sequence is a candidate only if each of its (k - 1)-length subsequences is a sequential pattern found at the $(k - 1)^{th}$ pass. A new scan of the database gathers the support for each candidate sequence and finds the new set of sequential patterns. This set becomes the beginning for the next pass. This continues until the candidate sequence generation stops or no frequent sequences can be found after the end of a scan.

There are two main constituents of the Generalized Sequential Pattern mining are as follows:

1. Candidate Generation
2. Support Counting

Candidate Generation: In the candidate generation process, candidate k-sequences are generated based on (k-1) sequences. There are two steps in candidate generation.

1. Join Phase
2. Prune Phase

These steps are based on contiguous subsequence concept.

Contiguous Subsequence: Given a sequence s = ($s_1 s_2$ ... s_n) and subsequence c, c is a contiguous subsequence of s if any of the following conditions hold:

a.) c is derived from s by dropping an item from either s_1 or s_n.

b.) c is derived from s by dropping an item from an element s_j that has at least 2 items.

c.) c is a contiguous subsequence of c_i, and c_i is a contiguous subsequence of s.

Join Phase: Candidate k-sequences are generated by joining two (k-1)-sequences that have the same contiguous subsequences, when we join the two sequences the item can be inserted as part of the element or as a separated element.

Prune Phase: Those candidate sequences that have a contiguous subsequence whose support count is less than the minimal support are deleted.

Support Counting: In the process of passing over the data, each data sequence is read at a time, and the counter of the support value increases at the same time.

Constraints

Although efficient algorithms have been proposed, mining a large amount of sequential patterns from large sequence databases is an inherently and computationally expensive task. For effectiveness and efficiency considerations, constraints are essential in many data mining applications. Applying constraints on different stages of the mining process can improve the efficiency.

According to Morzy et al. (2002), techniques applicable to constraint-driven pattern discovery can be classified into three groups.

- Post-processing (filtering out patterns that do not satisfy user-specified pattern constraints after the actual discovery process);
- Candidate filtering (application of pattern constraints to reduce the number of processed candidates);
- Dataset filtering (restricting the source dataset to objects that can possibly support patterns that satisfy user-specified pattern constraints.)

The constraint applied on a sequential pattern (α) can be written as $C(\alpha)$. Constraint based sequential pattern mining requires the user to identify the entire set of patterns which comply with that constraint. Constraints can be studied from different points of view such as from the application perspective and the constraint pushing perspective. From the application perspective there are seven types of constraints (Pei et. al, 2002).

Types of Constraints

Constraints can be classified into 7 major types based on the application point of view.

1. Item Constraint: Specifies the items, which should or should not be included in that particular pattern. It is represented as

   $C_{item}(\alpha) \equiv (\propto i : 1 \leq i \; len(\alpha), \alpha[i] \; \theta \; V)$,

   where V = subset of the various items, $\propto \in \{\forall, \exists\}$ and $\theta \{\subseteq, \not\subseteq, \supseteq, \not\supseteq, \in, \notin\}$

2. Length Constraint: Indicates the pattern length, which can be the number of times the items appear or the count of transactions executed. They can be represented as a maximum value of items or minimum value of items in the set.

   $C_{len}(\alpha) \equiv (len(\alpha) \; \theta \; t)$,

   where $\theta \in \{\leq, \geq\}$ and t is an integer.

3. Super pattern Constraint: Aims to find patterns that contain a particular set of patterns as sub- patterns. It can be represented as

   $C_{pat}(\alpha) \equiv (P \sqsubseteq \alpha)$,

   where P is a set of patterns.

4. Aggregate Constraint: It is applied on a collection of items in the given pattern. The aggregation function can be sum, standard deviation, average, maximum value, minimum value etc.

5. Regular Expression Constraint: It is applied in the form of an expression ranging over items defined in the set. It uses expression operators such as Kleene closure and

disjunction. The function can be considered as workable only if it is acceptable by a equivalent automata.

6. Duration Constraint: Applicable for sequence databases with transactions having a time stamps for every sequence. The difference in time stamps between the initial and final transactions should be higher or lower than a specified duration. This can be represented as ;

   $C_{dur}(\alpha) \equiv Dur(\alpha)\ \theta\ \Delta t$

   where $\theta \in \{\leq, \geq\}$ and Δt is an integer.

   Maximum span and window-size constraints are types of duration constraint.

   Maximum Span: It is the maximum allowed time difference between the latest and earliest occurrences of items in the entire sequence.

   Window Size: It is the maximum allowed time difference between the latest and earliest occurrences of items in any *itemset*.

7. Gap Constraint: Applicable for sequence databases where transactions have time stamps for every sequence as well. The pattern needs to present itself regularly and follow the condition that the difference of timestamps between every adjoining transaction stays longer or shorter than a specified gap. It can be represented as

   $C_{gap}(\alpha) \equiv Gap(\alpha)\ \theta\ \Delta t$,

   where $\theta \in \{\leq, \geq\}$ and Δt is an integer.

   Maximum gap and minimum gap constraints are examples of Gap Constraints.

Maximum Gap: It is the maximum allowed time difference between the latest occurrence of an item in an itemset and the earliest occurrence of an event in its immediately preceding itemset.

Minimum Gap: It is the minimum required time difference between the earliest occurrence of an item in an itemset and the latest occurrence of an item in its immediately preceding itemset.

According to Pei and Han (2002), among the constraints listed above time constraints are support related. To find whether a sequential pattern satisfies these constraints the sequence database should be examined. For other constraints, whether the constraint is satisfied can be determined by the frequent pattern themselves, without referring to the support counting process.

CHAPTER 4

DEVELOPMENT OF THE CONSTRAINT BASED SEQUENTIAL PATTERN MINING (CBSPM) TOOL

In this chapter, stages of the development of a sequential pattern mining tool which is named as Constraint Based Sequential Pattern Mining (CBSPM) is presented. The algorithm, database design, Graphical User Interface (GUI) of CBSPM and a utility for generating artifical datasets, are explained in detail.

General Characteristics of CBSPM Tool

In traditional sequential pattern mining techniques, the main purpose is to discover sequences that are frequent, meaning that the number of sequences which contains the subsequence that user interested in a database, exceeds user specified minimum support. There are two primary problems which user may encounter while using these traditional mining techniques. Firstly, the users will be swamped with a huge number of frequent patterns, most of which are useless for their purposes. Secondly, as the size of the dataset becomes larger, obtaining the complete set of patterns would require a lot of processing power. Constraint based data mining technique is an efficient solution to overcome these problems as it restricts the patterns within a set of conditions based on the requirements of the user. Although CBSPM is an essential task of data mining, in the literature there is no such a data mining tool that handles most of these user-specified constraints simultaneously (*Item Constraint, Length Constraint, Super pattern Constraint, Aggregate Constraint, Duration* and *Gap Constraint)*.

CBSPM is designed as a new tool which handles all of these user-specified constraints. It is developed by using JAVA programming language. CBSPM also

provides ease of use with its graphical user interface (GUI). In addition, a Data Generation facility is available in CBSPM.

The dataset and constraints can be selected by the user in CBSPM. It is necessary for the user to define the minimum support threshold but time constraints and other user specified constraints are optional. If the user is not interested in time constraints, default values (*min-gap*=0, *max-gap*=9999, *window-size*=1 and *maximum-span*=9999) will be assigned. This means the absence of time constraints. For time constraints, window-size is defined to specify the items which are in the same element. Items occur within a time difference which is smaller than window-size can be taken as in the same element and the order of items doesn't matter in an element. Maximum gap and minimum gap are defined to specify the gap between any two adjacent elements in the sequence. If the time difference is not in the range between maximum gap and minimal gap then this two cannot be taken as two consecutive elements in a sequence (Fiot et. al, 2007).

CBSPM is designed to discover the frequent patterns in a given dataset by satisfying the following constraints: *user-defined minimum support, user-defined min-gap, max-gap, window-size and maximum span time constraints and user defined Length Constraint, Super pattern Constraint, Aggregate Constraint.* There are two essential techniques in CBSPM algorithm. The post-processing technique and candidate filtering technique which are described in chapter three are used together in the development of CBSPM.

Firstly the algorithm requires the user to select the database and to specify minimum support, time constraints and the additional constraints as inputs. In the first step of the CBSPM algorithm, minimum support and time constraints are handled in

binaryGenerator(), search() and setSequenceObject() methods. Candidate (C) and frequent (L) sequences are formed in these methods. After finding C_2(2-length candidate sequences) by combining L_1(1-length frequent sequences) in binaryGenerator(), L_k where $k \geq 2$, is formed in setSequenceObject() method. setSequenceObject() takes the candidate sequences from C and then counts it in the array list (SD) which has all the sequences in the dataset. While searching setSequenceObject() method also checks that whether the sequence satisfies the time constraints or not and whether the count of that sequence is greater than the minimum count or not. If both of them are satisfied than the candidate sequence is added to L as a frequent sequence. In the final step a post-processing step, filtering out patterns that do not satisfy the additional constraints (*Item Constraint, Super-Pattern Constraint, Length Constraint* and *Aggregate Constraint*) is applied. The algorithm terminates by displaying the frequent patterns which are filtered according to user specified constraints. Figure 2 shows the algorithm of CBSPM.

```
Input  : (1) S= a sequence dataset
         (2) Min Sup : a support threshold
         (3) a mingap, maxgap, window-size, maximum gap time constraints
         (4) item, length, super pattern, aggregate user-specified constraints

Output: frequent sequences satisfying constraints

Find different items and add items to C1
Find L1 (set of 1- sequence which is greater than mincount)

C2= binaryGenerator(L1);  // generate new candidate 2-length sequences
//in setSequenceObject() method
for all candidates c ∈ C2 do
 if c satisfies user specified time constraints
                c.count++
      L2= {c ∈ C2| c.count ≥ mincount}
 end if
end;
     for (k=3;Lk-1≠Ø;k++) do
          for all sequences l ∈  Lk-1
               Ck = combine (l)
               if  Ck = Ø then break;
               forall candides c ∈  Ck do
                    if Lk-1 contains subsequence of Ck and
                         c satisfies user – specified time constraints
                         c.count++
                    end if
                    Lk={c ∈   Ck | c.count ≥ mincount};
               end
     end;
     for all frequent sequences l ∈  L do     // L ⊆ Lk
          if S doesn't contain l which satisfies user-specified constraints then
      delete l from L
          end if;
  end;
    output patterns from L satisfying all constraint
```

Fig. 2 Algorithm of CBSPM

Database Structure of CBSPM

The sequence dataset S has two tables with the names “Sequence” and “Item Price”. Sequence table is formed by a set of records (sid, s), where s is a data sequence and *sid* is the identifier of this data sequence Each column, s, in a sequential dataset holds a particular sequence, which consists of ordered list of items with transaction time of those items. s column is represented as $(a_1\text{-}t_1; a_2\text{-}t_2; a_3\text{-}t_3;\ldots\ldots; a_n\text{-}t_n)$ where a_j is an item $1\leq j\leq n$, and t_j stands for the time at which a_j occurs, $t_{j-1}\leq t_j$ for $2\leq j\leq n$. The data value of a_j is String and type of t_j is integer. Figure 3 is an example table of CBSPM dataset.

Sid	S
1	(a-2;b-3;a-4;c-6;e-10;e-12)
2	(d-5;a-7;b-7;e-7;d-9;e-9;c-14;d-14)
3	(b-15;c-17;f-17;e-18;b-22;c-22)

Fig. 3 A sequence dataset

Item Price table is formed by a set of records (item, price). Figure 4 demonstrates an example *Item Price* table.

Item	Price
A	5
b	8
C	3
…	..
f	7

Fig. 4 An item-price table

Classes and Methods of CBSPM

In this study, CBSPM is developed by using the Java programming language. The class diagrams are extended with IBM Rational Rose. Figure 5 depicts the constraint based generalized sequential pattern classes and their relationships.

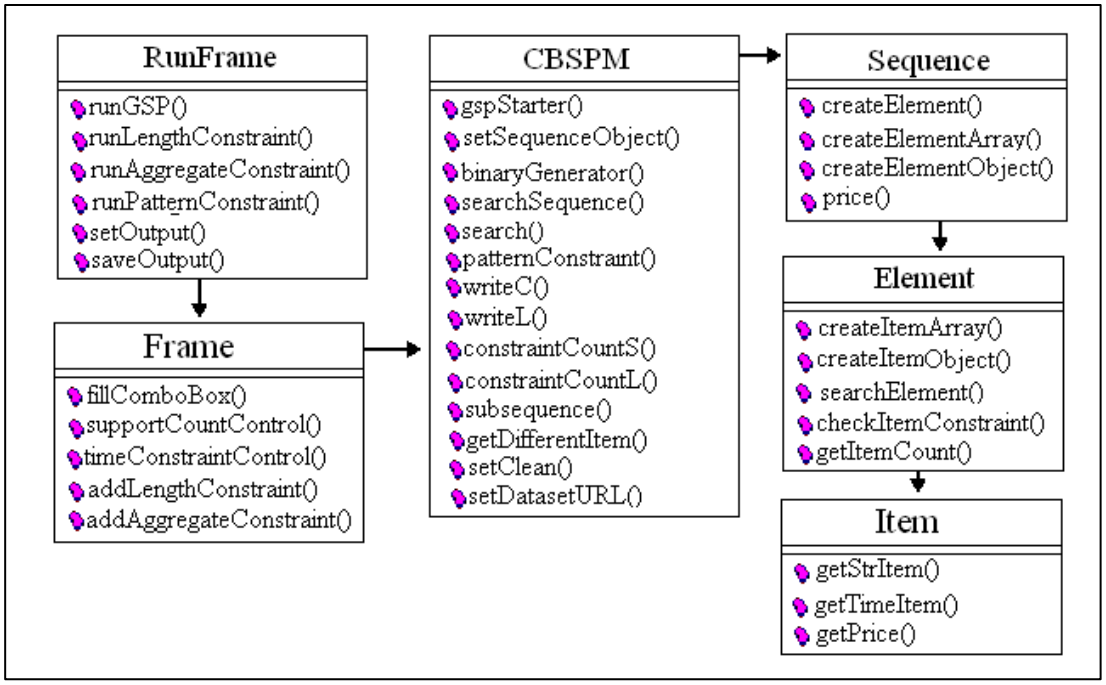

Fig. 5 Class structure of CBSPM

CBSPM is the most essential class that handles the constraints. Runframe class is used for invoking a frame object and Frame class is used as a gateway for users to indicate the constraints and to invoke the CBSPM object.

There are two essential steps in CBSPM algorithm. First one is the generation of candidates. This is divided into two phases, Join Phase and Prune phase. Second one is counting the candidates being generated. Both of these are performed in setSequenceObject () and in binaryGenerator() methods. These are the core of the CBSPM class. All of the classes and methods in CBSPM are presented below.

CBSPM Class

gspStarter(mincount,constraints): This method receives min count and other constraints from the user and invokes the CBSPM object.

setSequenceObject(): This method finds different items and creates the array of candidate sequences (C) here. After that it adds different items to C_1. It calls the search() method and finds the frequent items which has a number of value greater than minimum support. Then it creates the frequent items array (L) and adds frequent items to L.

binaryGenerator(): This method takes L_k from L than calls the *subsequence* method if the value which return from *subsequence* is true, it combines L_k and adds them to $C_{k+1.}$ Then it calls the *search* method. The *search* method receives C_{k+1} as its parameters and checks if C_{k+1} satisfies min_support and time constraints. If C_{k+1} satisfies the constraints *binaryGenerator* adds C_{k+1} to $L_{k+1.}$ This process continues until no frequent sequences left.

searchSequence(sequence,sequnce): This method receives two sequence object as its parameters and checks if the second one is a subsequence of the first one.

search(sequence): This method receives a sequence as its parameter. It takes items and their transaction times in that sequence and checks whether this sequence handles time constraints or not.

patternConstraint(): This method controls that if the sequences in L satisfies the Item and Super Pattern constraints.

writeC(): This method prints C as an output.

writeL(): This method prints L as an output.

constraintCountS(): This method controls that if the sequences in L satisfies the Length constraint. If the number of item is greater than the value which is given by the user it deletes that sequence from L.

constraintCountL(): This method controls that if the sequences in L satisfies the Length constraint. If the number of item is smaller than the value which is given by the user it deletes that sequence from L.

subsequence(): This method checks the contiguous subsets.

getDifferentItem(): This method finds different items and returns them.

setClean(): This method initializes all arrays and variables in CBSPM.

setDatasetURL(): This method finds the path of the Dataset which is selected by the user and send it to setDataFileName method as a parameter.

Fig. 6 Methods of the CBSPM Class

Sequence Class

createElement() : This method creates an element array.

createElementArray(n): This method creates an element array with a length of n and it assigns an element object to every element of the array.

*createElementObject(array):*This method receives an array as its parameter and creates an element object.

*price():*This method controls that if the sequences in L satisfies the Aggregate constraint.

Fig. 7 Methods of the Sequence class

Element Class

createItemArray(itemArray): This method creates an item array with a length of *n* and it assigns an item object to every element of the array.

createItemObject(array): This method receives an array as its parameter and creates an item object.

getElement(): This method returns the name of the element.

searchElement(Element e): This method receives an element as its parameter. It checks if an element contains e or not..

checkItemConstraint(item): This method checks if L consists of *item* or not.

getItemCount(): This method returns the item count of the element.

Fig. 8 Methods of the Element class

Item Class

getStrItem() : This method returns the name of an item.

getTimeItem(): This method returns the transaction time of an item.

getPrice(): This method returns the price of a item.

Fig. 9 Methods of the Item class

Frame Class

fillComboBox(): This method takes different items from dataset and fills the combo box with these items.

supportCountControl(): This method checks if the support value is entered or not.

timeConstraintControl(): This method checks if the time constraints are entered or not.

addLengthConstraint():This method checks if the length constraint is selected or not.

addAggregateConstraint():This method checks if the aggregate constraint is selected or not.

Fig. 10 Methods of the Frame class

RunFrame Class

runGSP(count, windowsize, mingap, maxgap, superPattern): This method receives *count, windowsize, mingap, maxgap, superPattern* as its parameters and sends these parameters to *gspStarter()* method.

runLengthConstraint (choice, length): This method receives the choice of the user and length value as its parameters and sends them to the CBSPM.

runAggregateConstraint(choice1,choice2, price): This method receives the choice of relational operator, choice of aggregation function, and price value as its parameters and sends them to the CBSPM.

*runPatternConstraint(pattern):*This method receives *super pattern* value as its parameters and sends them to the CBSPM.

setOutput(): This method prints the output to the text area in the frame.

saveOutput(): This method saves the output.

Fig. 11 Methods of the RunFrame class

The full source-code of the program is available in the CD.

Data Generator

Data Generator facility is designed for generating artificial datasets in addition to provide ease of use to the users.

Datasets of sequences containing items and transaction times of items are generated by *DataGenerator*. The input parameters to *DataGenerator* include the item list, the number of sequences, and number of items in a sequence, the items which will be in a sequence, transaction time of items. There is a default item array which has 10 items in *Data Generator* class. When a *DataGenerator* object is invoked, first the number of sequences is generated randomly from a Uniform distribution. This number shows that how many sequences will be in *Sequence dataset*. After defining the number of sequences, *Data Generator* starts to form the sequences. There are three important parameters while forming the sequences. These are the number of items in a sequence, items which will be in that sequence and transaction times of items. Number of items in a sequence is selected randomly from a Uniform distribution. The user assigns the limits of the interval of this Uniform distribution. After the number of items in a particular sequence is determined, the items should be assigned to that sequence. First the items should be defined. A random index is selected from a Uniform distribution in the interval [0,"length of item array"]. After this random variable is acquired, the item in the item array with that index is assigned to the sequence. Afterwards transaction time should be allocated to each item. As soon as an item is assigned to a sequence, transaction time of that item should be acquired. The transaction time of an item must be equal or greater than the transaction time of the previous item. In order to provide each item with an appropriate transaction time, random numbers for each item are

generated from Uniform distribution in the interval [0,'user defined time limit']. These numbers are sorted in an ascending order and then they are assigned to each item.

Graphical User Interface (GUI)

CBSPM is a new sequential pattern mining tool which is not only uses the support threshold but also handles the user-specified time and other constraints (*Item, Super Pattern, Length* and *Aggregate*). It has a user-interface that provides ease of use to users. Figure 12 is the opening window of the CBSPM.

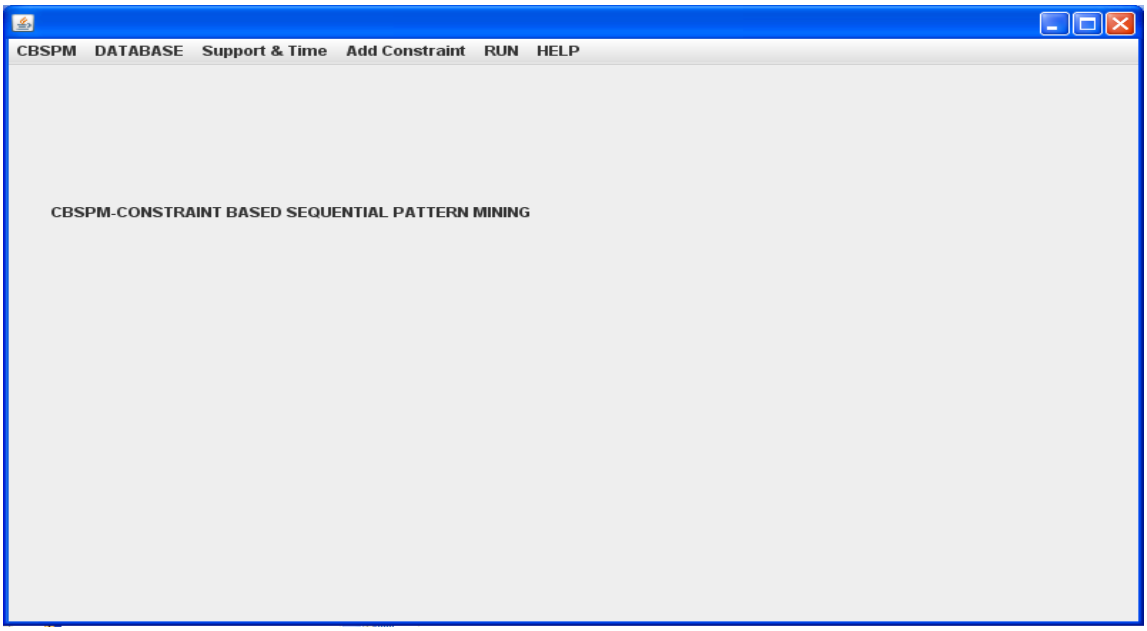

Fig. 12 The Opening Window of CBSPM

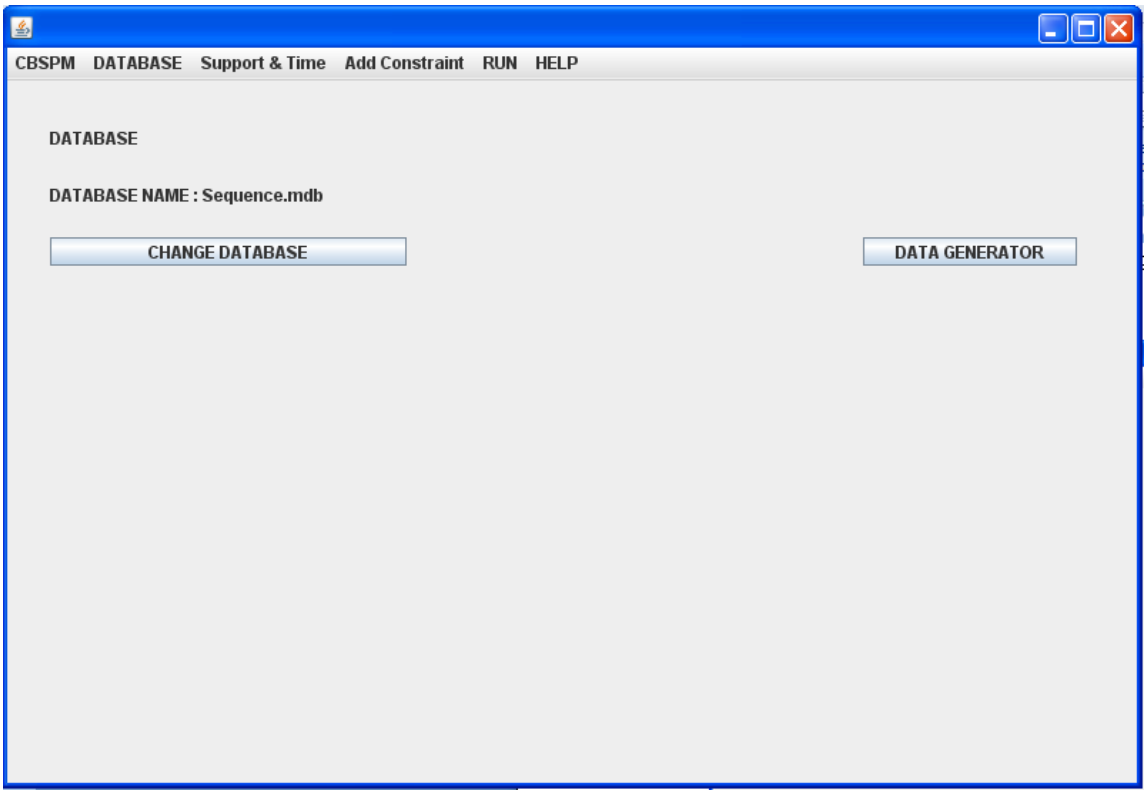

Fig. 13 Dataset Window

Figure 13 shows the Dataset window. First, the dataset should be selected. There is a default sequence dataset which is used by CBSPM.

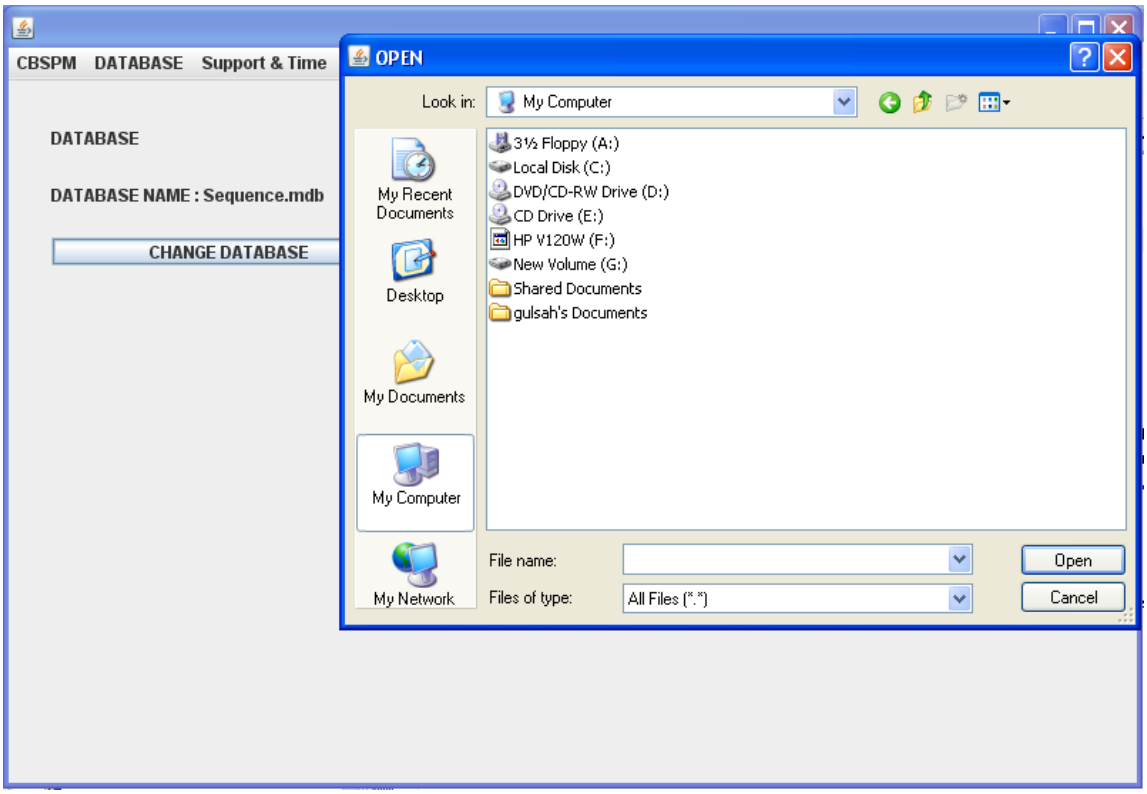

Fig. 14 Change Dataset Window

The dataset can also be changed by the user as shown in Figure 14. Moreover a dataset can be generated by clicking the Data Generator button.

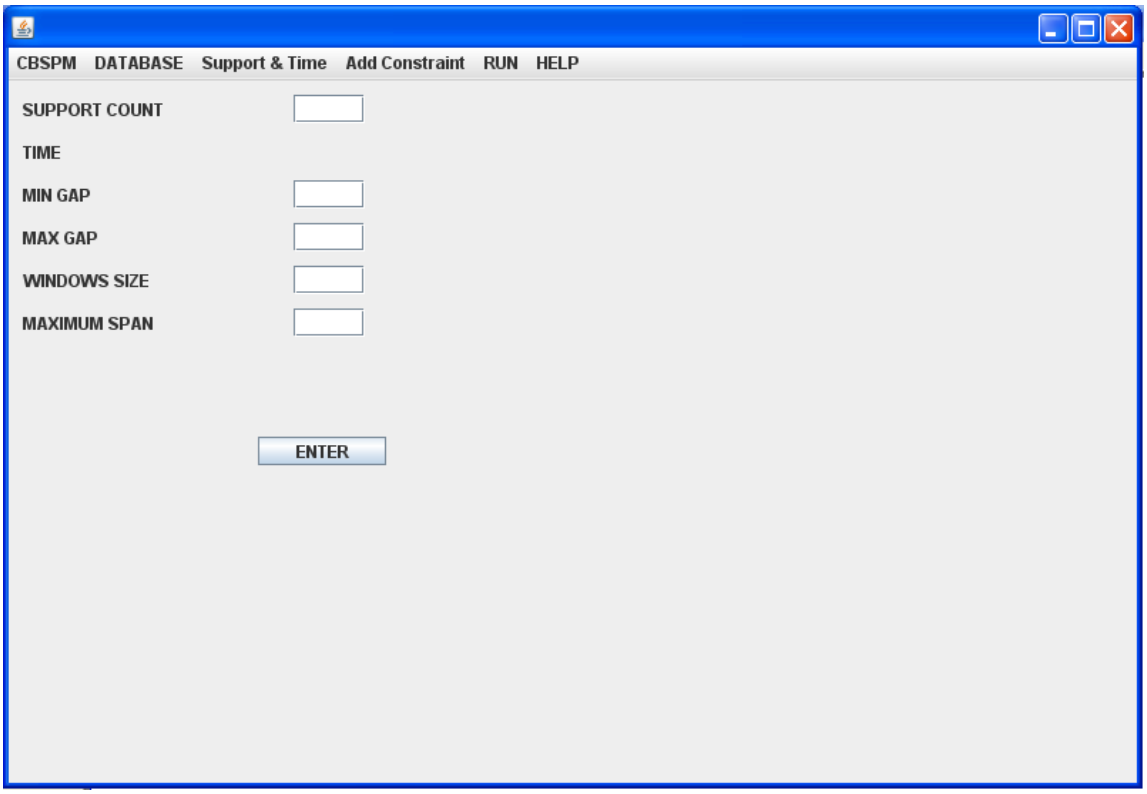

Fig. 15 Support & Time Constraints Window

After selecting the dataset the support count should be entered. As shown in Figure 15 there are text fields for minimum support threshold and time constraints in the Support&Time panel. If the support count field is empty CBSPM will warn the user, but the time constraints are optional. If the time constraints weren't entered, CBSPM will assign default values (*minimum gap*=0, *maximum gap*=9999, *window size*=1 and *maximum span*=9999).

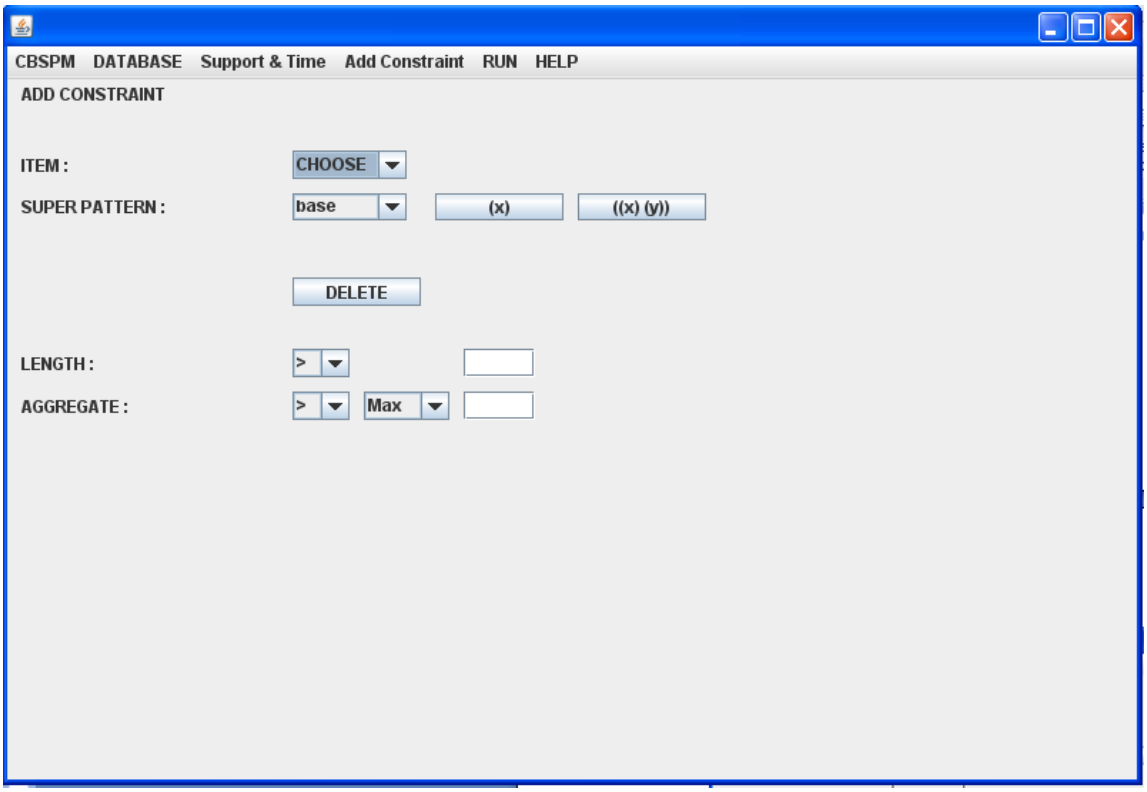

Fig. 16 Additional Constraints Window

After selecting the support count and time constraints the Support & Time panel can be selected from the top menu for adding the other constraints. Figure 16 demonstrates the additional constraints window. There are four different constraints and these are the optional constraints.

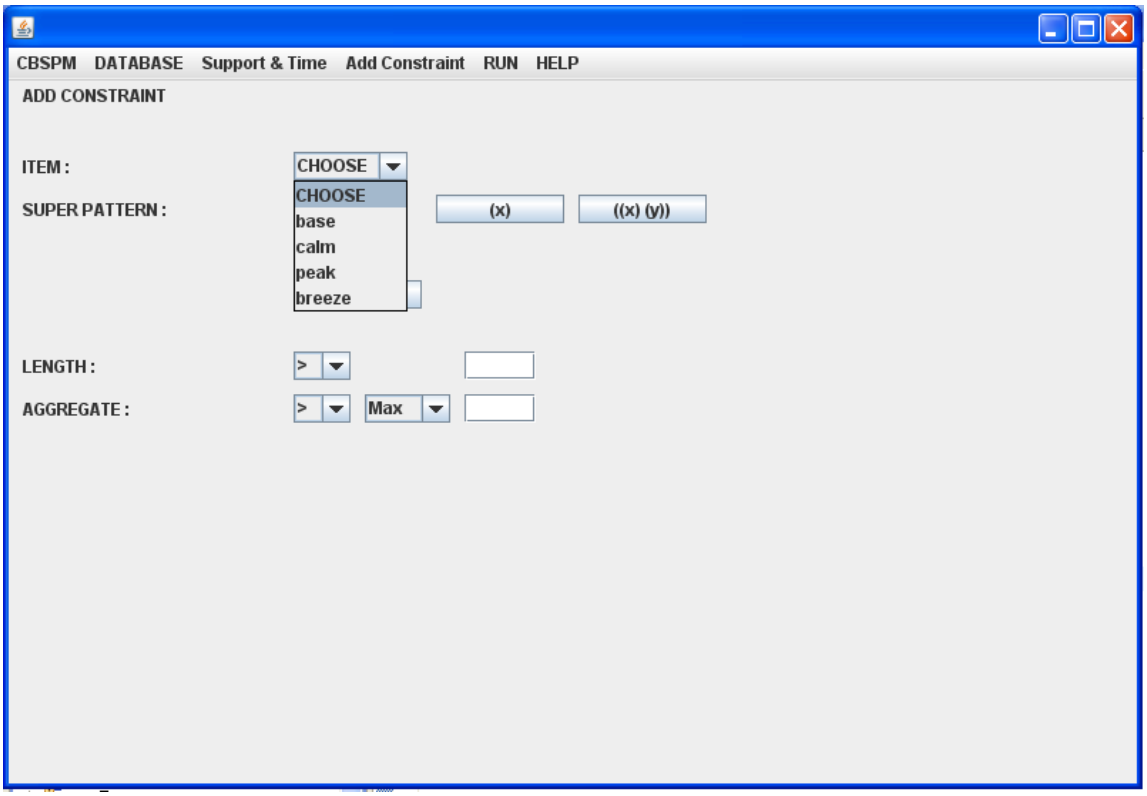

Fig. 17 Item Constraint

The first constraint is the Item constraint. The item for which the user wants to find the frequent sequences that include this item can be chosen and the (x) button can be pressed. This button demonstrates that this is a single item. Figure 17 demonstrates the Item Constraint Window.

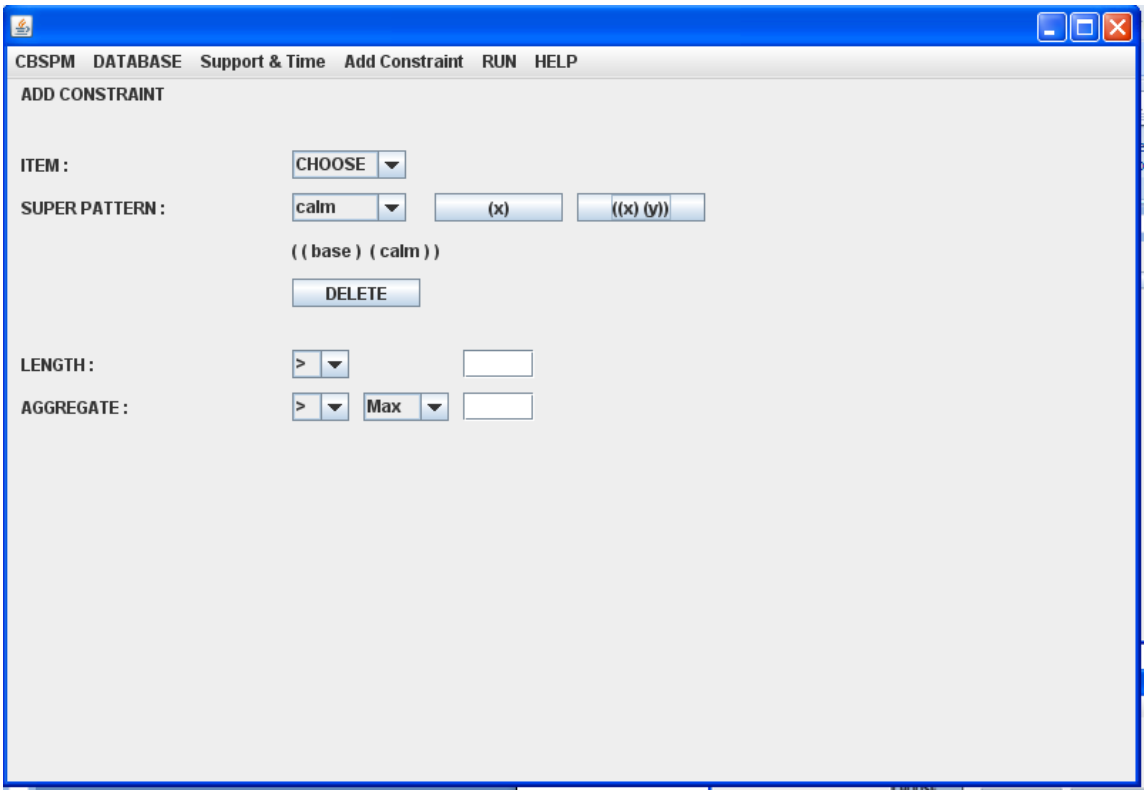

Fig. 18 Super Pattern Constraint

Another additional constraint is the super pattern constraint. If the user is interested in a super pattern, which is defined within a window-size for example (base-calm), then first base must be selected from the combo box and the (x) button which identifies a single element must be clicked. This button is for adding a single item. After adding the base, calm should be selected from the super pattern combo box and the {(x)(y)} button that identifies an element which has more than one item should be clicked. Super Pattern Constraint window is shown in Figure 18.

If the user is interested in finding a pattern with two or more items that align respectively like (base, calm, base). First the base should be selected and (x) should be clicked, then the calm should be selected and (x) should be clicked. Lastly, the base should be selected again and (x) should be clicked.

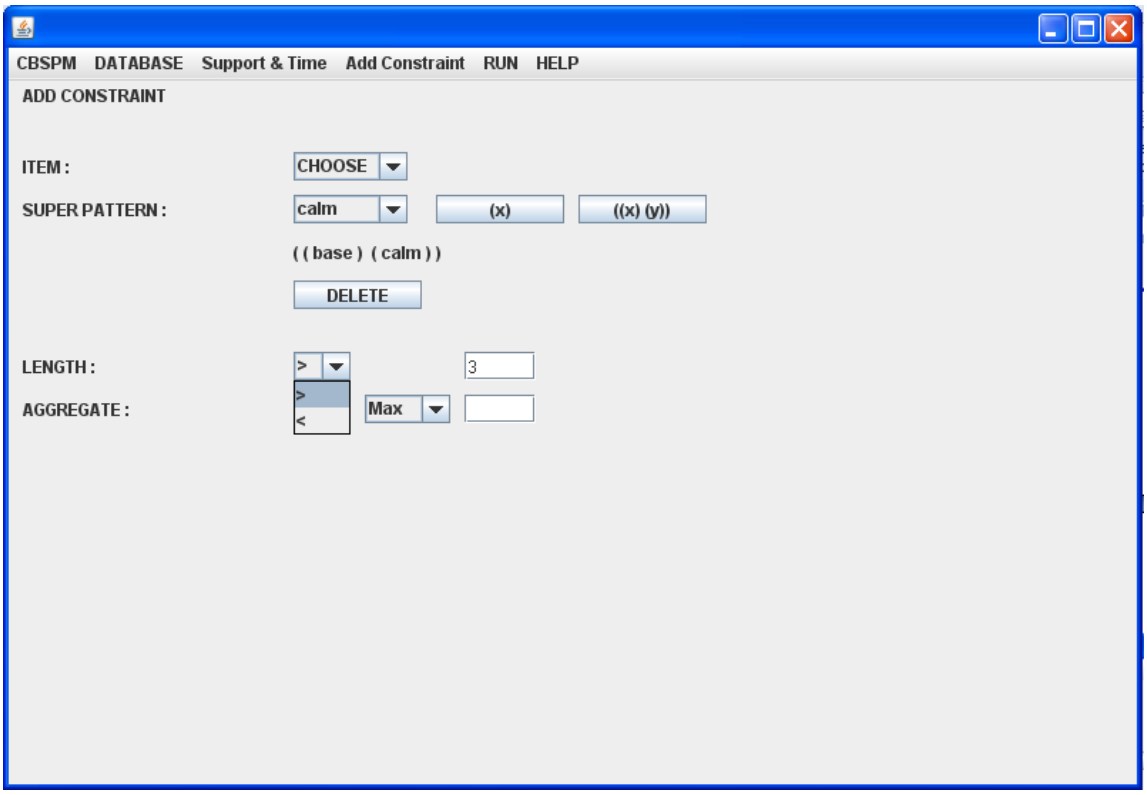

Fig. 19 Length Constraint

Figure 19 shows the Length Constraint. If the user wants to indicate the pattern length the length constraint can be defined. There is a combo box from which the user can select the relational operator (≤, ≥). After selecting the operator the length should be entered to the text box. For example, if the user selects the sign" ≥" and writes 3 to the text field, this means that the user interested in with the frequent sequences which has number of items greater than or equal to 3.

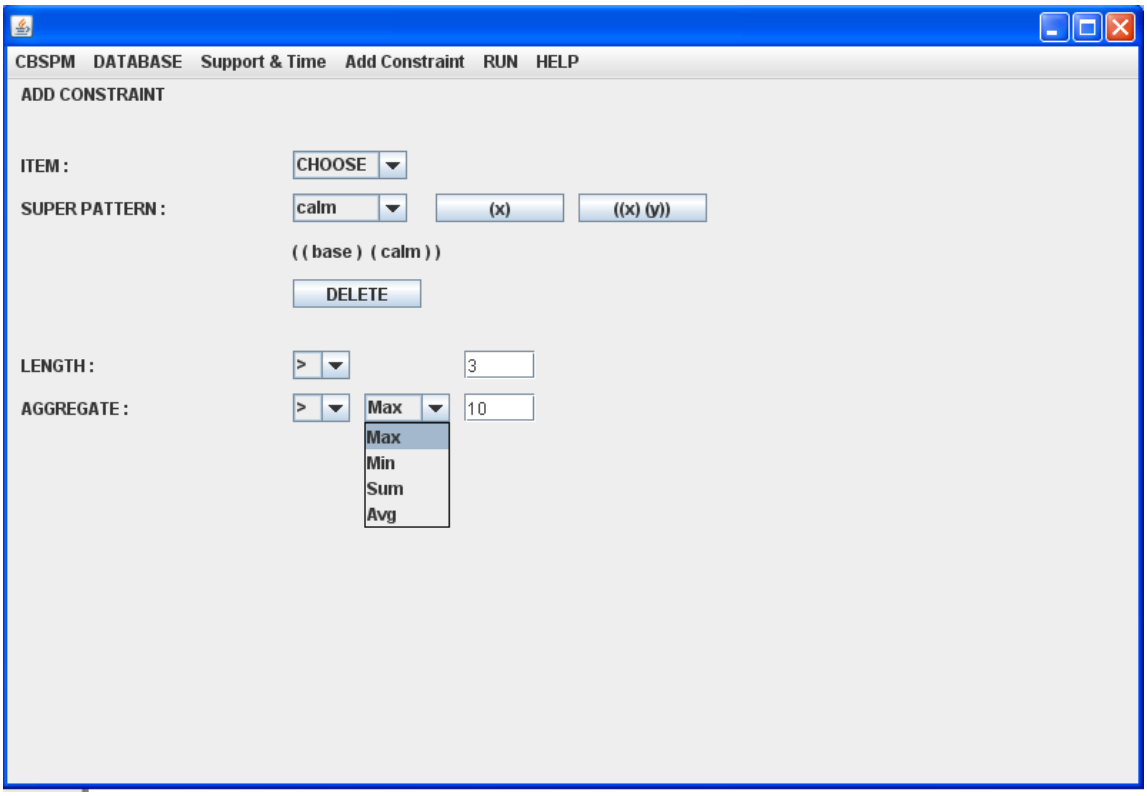

Fig. 20 Aggregate Constraint

Aggregate constraint is shown in Figure 20. If the user is concerned with the price of a transaction, for instance if the user wants to find the sum of prices which is greater or less than or equal to a value, this constraint should be used. Similar to length constraint there is a relational operator combo box. There is an additional combo box for aggregate functions. Four different functions can be selected from combo box. These are max, min, sum and avg. After selecting one of these functions, the value of price should be entered to the text box. For example if "≥" and sum are selected from combo boxes and 30 is entered to the text box this means that the user is interested in the transactions which are frequent and the sum of the prices in that transaction is greater than or equal to 30.

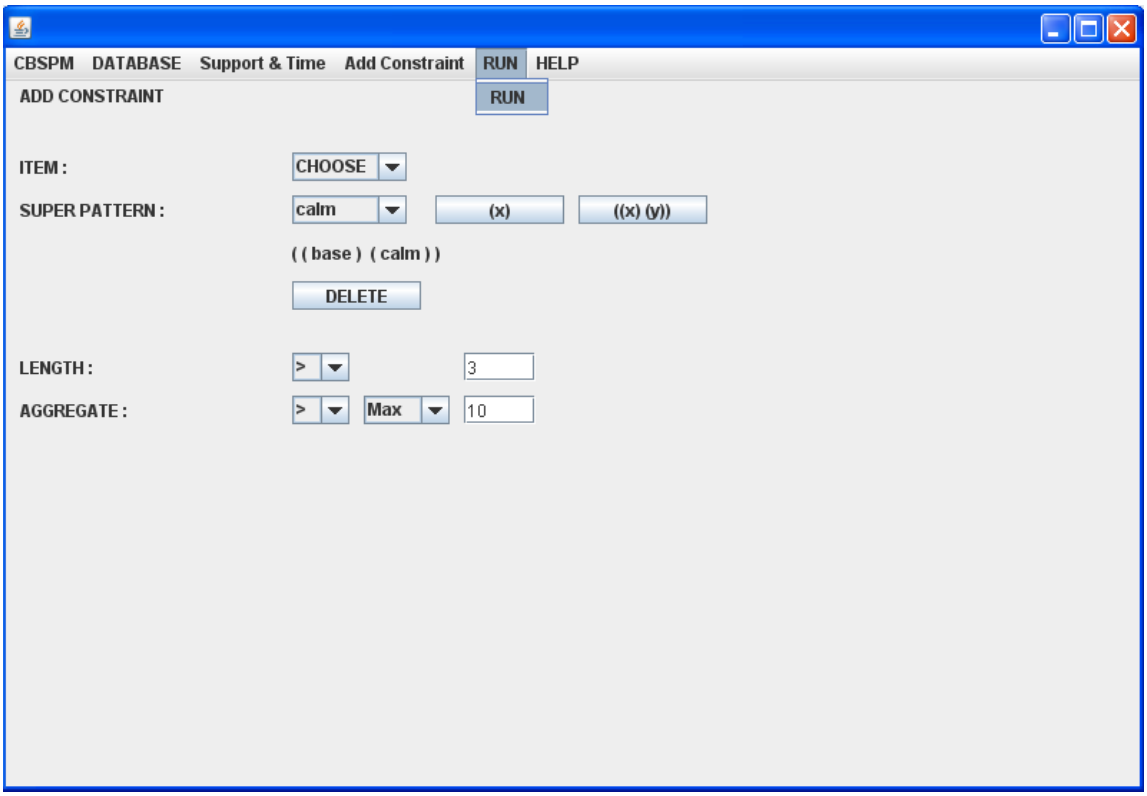

Fig. 21 Run Window

After selecting the support count and the other constraints, run button should be selected from the top menu. When the run button is pressed a new frame which is shown in Figure 21 is opened and the results are shown there. If the user wants to save the results the output can be saved by pressing the save button. If the user wants to exit from the output page, the close button should be clicked.

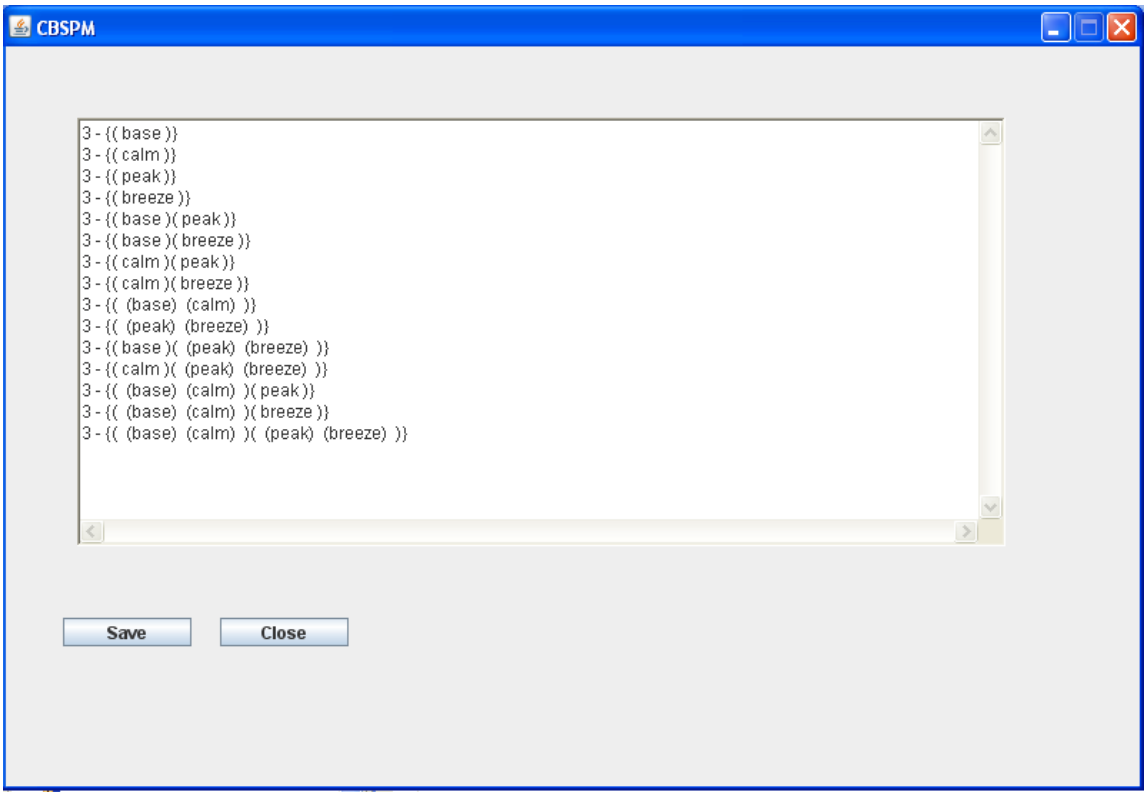

Fig. 22 Output Window

In the results window which is shown in Figure 22, the first column defines the counts of the frequent sequences. The second column shows the frequent sequence. Curly brackets signify a sequence whereas parentheses indicate items. For example (base) is an item. If parentheses are nested the outer set of parenthesis implies an element, for example ((base)(calm)) stands for an element that consists of 2 items. In the last line of the result window { ((base) (calm)) ((peak) (breeze)) } , there is a sequence with 2 elements each of which contains 2 items.

CHAPTER 5

Evaluation of Performance of CBSPM

This chapter aims to explain the verification of CBSPM and to show the performance results of CBSPM.

All tests were conducted on a 2.8GHz Intel Pentium PC with 1024MB memory running on Microsoft Windows XP Professional Edition.

In order to ensure that it is working as it is suggested and to evaluate performance results of this tool several tests were conducted. For the verification of CBSPM the default GSP database of WEKA is used. For evaluating the performance results several synthetic datasets, which are generated by *DataGenerator* class of CBSPM, are used. Using different datasets six tests were conducted.

Verification of the CBSPM

Since WEKA is a Data Mining tool which includes GSP as a Sequential Pattern Mining Algorithm, CBSPM's output is compared with WEKA's output in order to verify the CBSPM. WEKA's dataset for GSP is used in both tools. In order to use the WEKA's dataset in CBSPM it is converted to CBSPM's dataset design. The default GSP database of WEKA is available in Appendix A. Since WEKA does not handle the constraints including time constraints the time constraints are fixed (infinite *max-gap* and *window span*, *min gap* equal to zero and *window size* equal to 1). After execution of WEKA and CBSPM, both of them have same 15 frequent sequences as an output. Figure 23 and Figure 24 shows the output of WEKA and output of CBSPM respectively.

```
GeneralizedSequentialPatterns
Total number of frequent sequences: 15
- 1-sequences
[1] <{power=base}> (3)
[2] <{power=peak}> (3)
[3] <{wind=calm}> (3)
[4] <{wind=breeze}> (3)
- 2-sequences
[1] <{power=base}{power=peak}> (3)
[2] <{power=base}{wind=breeze}> (3)
[3] <{power=base, wind=calm}> (3)
[4] <{wind=calm}{power=peak}> (3)
[5] <{wind=calm}{wind=breeze}> (3)
[6] <{power=peak, wind=breeze}> (3)
- 3-sequences
[1] <{power=base}{power=peak, wind=breeze}> (3)
[2] <{power=base, wind=calm}{power=peak}> (3)
[3] <{power=base, wind=calm}{wind=breeze}> (3)
[4] <{wind=calm}{power=peak, wind=breeze}> (3)
- 4-sequences
[1] <{power=base, wind=calm}{power=peak, wind=breeze}> (3)
```

Fig. 23 Output of WEKA for the verification dataset

Constraint Based Sequential Pattern Mining
Total number of frequent sequences: 15
- 1-sequences
3 - {(base)}
3 - {(calm)}
3 - {(peak)}
3 - {(breeze)}
- 2-sequences
3 - {(base)(peak)}
3 - {(base)(breeze)}
3 - {(calm)(peak)}
3 - {(calm)(breeze)}
3 - {((base) (calm))}
3 - {((peak) (breeze))}
- 3-sequences
3 - {(base)((peak) (breeze))}
3 - {(calm)((peak) (breeze))}
3 - {((base) (calm))(peak)}
3 - {((base) (calm))(breeze)}
- 4-sequences
3 - {((base) (calm))((peak) (breeze))}

Fig. 24 Output of CBSPM for the verification dataset

Performance Test

The execution time is used as a performance metric, which is defined as the time interval from the starting time and ending time of mining process. The input parameters include the item list, the number of sequences, and number of items in a sequence, the items which will be in a sequence, transaction time of items, time constraints and *min_support*.

Base Test (Test1)

Several datasets generated by *DataGenerator* are used while conducting the performance tests.

First a base dataset is generated. In the base dataset the number of sequences is fixed to 100 by setting the limits of Uniform distribution in the interval [100,100]. Also the maximum number of items in a sequence is assigned as 5. After these two parameters are defined the items and transaction time of each item are assigned. Transaction time is generated randomly from a uniform distribution in the interval [1,60]. In the base test the *min-sup* value is set to 3. *max-gap* and *window-span* are set to 999 , *window-size* is defined as 1 and *min-gap* is set to 0. Under these circumstances the execution time of CBSPM is 0.06 seconds.

Test 2- Comparison of Execution Times with Varying The Number of Sequences

In Test 2 the execution times of CBSPM while using different datasets with different number of sequences are compared. The first dataset is the Base dataset which was generated in Base Test. The other datasets are generated by giving the same values to the input parameters as it is in the base dataset except the number of sequences. Number of sequence parameter is set to 300,500,750 and 1000 respectively in the other datasets. Figure 25 shows the execution times with varying number of sequences. When the number of sequences increases the execution time increases . The lower the sequence number, the better the performance of CBSPM.

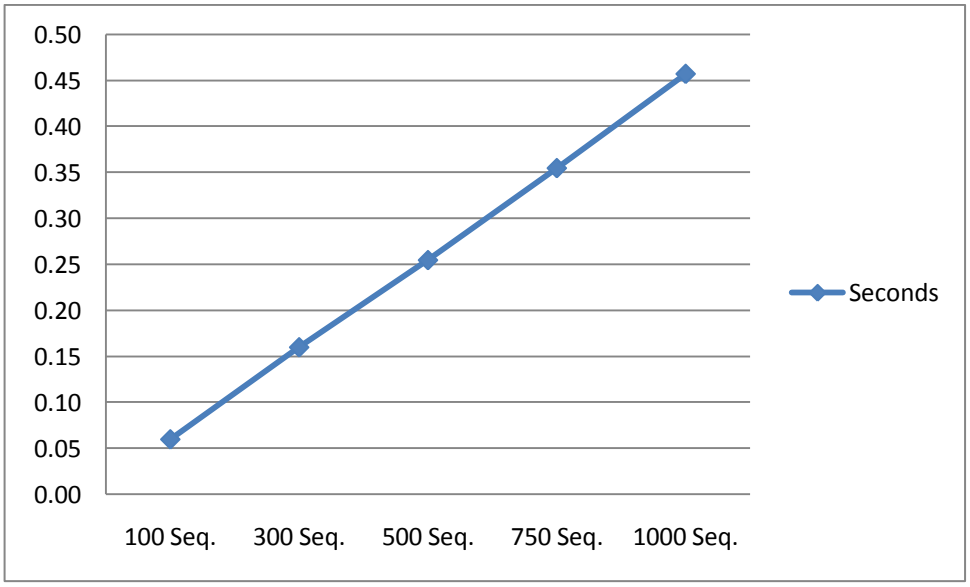

Fig. 25 Test 2- Execution times of CBSPM with varying the number of sequences

Test 3- Comparison of the Execution Times with Varying The Maximum Number of Items in a Sequence

In another series of tests, the influence of varying the maximum number of items in a sequence while the number of sequences, time constraints, and minimum support values are fixed as it is in the Base database is observed. The base dataset and other datasets which are generated by *DataGenerator* are used while comparing the execution times. When the maximum number of items in a sequence is high, CBSPM has longer execution times as shown in Figure 26. As long as the number of items per sequence does not exceed 10 CBSPM performs better. However, beyond 10 items per sequence CBSPM has longer execution times. While the number of items increases the number of outputs generated increases and this causes longer execution times.

Fig. 26 Test 3-Number of items in a sequence

Test4 – Comparison of Execution Times with Varying The Minimum Support

In test 4 the base dataset is used. Only minimum support value is changed and the other parameter are fixed as they are in base datasets. It is observed that CBSPM encounters problems when the minimum support threshold is low and time constraints are relaxed (large max-gap and window-span , small min-gap and window size) because of the huge number of candidates to be verified. Fig. 27 shows that increasing the minimum support threshold leads to shorter execution times. This is because fewer candidates have the potential to be frequent for higher values of minimum support.

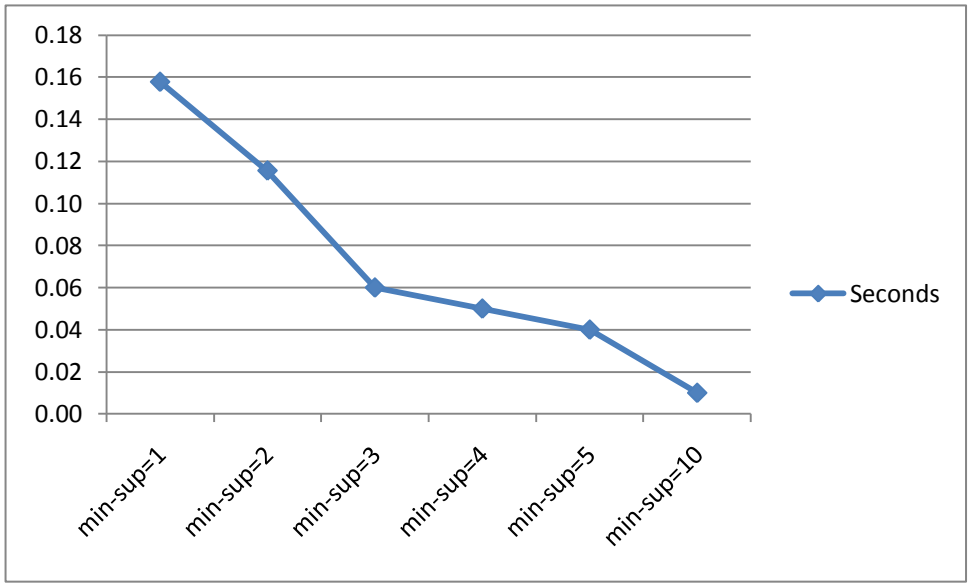

Fig. 27 Test 4- Execution times of CBSPM when the Minimum support is varied

Test 5- Addition of the user-specified constraints

Base test was conducted without constraints. In test 5 user-specified constraints were added respectively and the execution time of CBSPM was compared with the base test. Figure 28 shows that when the number of sequences, number of items in a sequence, number of different items and minimum support threshold are fixed the execution time is the longest without any constraints. Base model has the longest execution time. On the other hand CBSPM has better performance with respect to adding user-specified constraints (Item, Super Pattern, Length, Aggregate and Time) in a sequence database.

There are five steps in this test. In every step a constraint is added and the execution time is calculated. In the first step of Test 5, without any constraint the execution time is the longest. In the second step with adding only time constraints, the execution time is shorter this is because most of the sequences which could not handle the time constraints are eliminated. In the third, fourth and fifth steps the *item*, *length* and *super pattern* constraints are added respectively and in every step the execution times are shortened by the constraints.

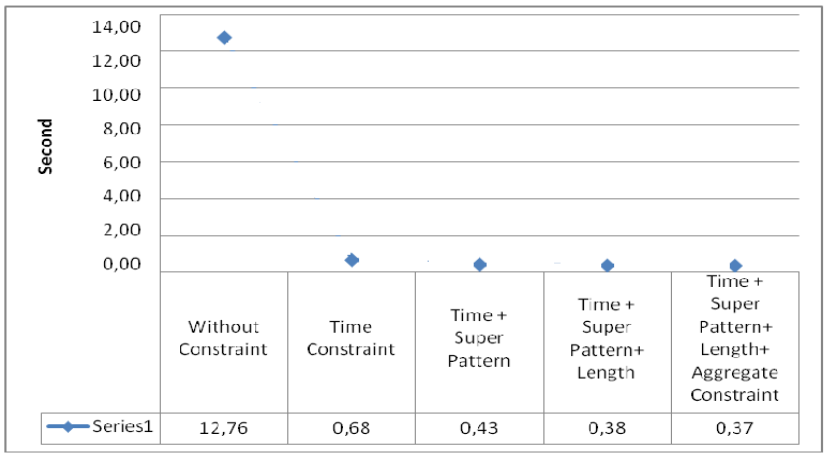

Fig. 28 Test 5-Addition of the User Specified Constraints

CHAPTER 6

CONCLUSION

Sequential pattern mining is an important data mining problem with wide applications. The information acquired from sequential pattern mining can be used in marketing, medical records, sales analysis. However, there are problems which arise in frequent pattern mining since the mining may have to generate or examine a combinatorially unstable number of transitional subsequences. GSP is a previously developed sequential pattern mining algorithm that identifies the sequential patterns with a minimum support value specified by the user. This approach may not be efficient in mining large sequence databases having numerous patterns or long patterns. Moreover, a user may be interested in some special frequent patterns. Constraint based data mining technique is a suitable solution to this problem since it facilitates obtaining filtered results according to a set of conditions based on the expectations of the user.

In this study a constraint based sequential pattern mining algorithm and based on this algorithm a constraint based sequential pattern mining tool (CBSPM) are developed for handling most of the user-specified constraints in sequential pattern mining. There are seven types of user-specified constraints. These are *Item Constraint, Length Constraint, Super pattern Constraint, Aggregate Constraint, Duration* and *Gap Constraint*. CBSPM algorithm is an extension of GSP algorithm. GSP uses candidate filtering technique for reducing the number of processed candidates. CBSPM also uses candidate filtering technique while handling minimum support and time constraints. In addition to candidate filtering technique CBSPM uses post-processing technique while handling the other constraints. This improves the efficiency of CBSPM.

The graphical user interface allows users to input parameters and to select the database. Furthermore a data generator facility is developed for generating artificial sequential datasets.

CBSPM was verified by using the same dataset with WEKA. WEKA's output and CBSPM's output were compared. The outputs of these tests show that CBSPM has same output with WEKA under the same conditions. After the verification of CBSPM, six performance tests were conducted in which the input parameters are changed

The performance test results show that the execution time of CBSPM increases linearly with the increase of the number of sequences. Moreover, maximum number of items in a sequence is an important parameter. When the maximum number of items in a sequence is high, CBSPM has longer execution times. In addition, it is observed that CBSPM has longer execution times when the minimum support threshold is low and time constraints are relaxed because of huge number of candidates to be verified. Without any constraint the execution time of CBSPM is the longest. By adding constraints, it can be seen that the execution time decreases.

This study can be extended in a few directions as a future work. One direction is adding the Regular Expression constraint to the algorithm. Another direction is adding the CBSPM algorithm to existing tools as a patch and CBSPM can be transformed as a web service.

APPENDICES

A. GSP Database of WEKA

```
@relation sequential_test_set
@attribute day {1, 2, 3}
@attribute time NUMERIC
@attribute 'power consumption' {power=base, power=peak}
@attribute 'wind speed' {wind=calm, wind=breeze}
@data
1,wind=calm
1,power=peak,wind=breeze
2,power=base,wind=calm
2,power=peak,wind=breeze
3,power=base,wind=calm
3,power=peak,wind=breeze
```

B. Contents of the CD

/CBSPM	JAVA Project folder of CBSPM and Data Generator

REFERENCES

Works Cited

Agrawal, R., and R. Srikant. "Fast Algorithms for Mining Association Rules." *In Proceedings of the twentieth International Conference on Very Large Databases.* 1994. 487-499.

Agrawal, Rakesh, and Ramakrishnan Srikant. "Mining Sequential Pattern." *In Proceedings of the Eleventh International on Data Engineering.* Taipei, Taiwan, 1995. 3-14.

Antunes, C., and A. Oliveira. "Generalization of Pattern-Growth Methods for Sequential." *In International Conference Machine Learning and Data Mining.* 2003. 239-251.

Chang, C., M. Lin, and S. Hsueh. "Mining closed sequential patterns with time constraints." *In Proceedings of workshop on software engineering, databases and knowledge discovery.* Taipei, Taiwan: International Computer Sysmposium, 2006. 33-46.

Chen, Yen L., and Ya H. Hu. "Constraint Based Sequential Pattern Mining: the consideration of recency and compactness." *Decision Support Systems*, 2006: 1203-1215.

Cooley, Robert., B. Mobasher, and J. Srivastava. "Web Mining: Information and Pattern Discovery on the World Wide Web." *In Proceedings of the of the nineth International Conference on Tools with Artificial Intelligence.* 1997. 558.

Fiot, C., A. Laurent, and M Teisseire. "Extended Time Constraints for Sequence Mining." *In fourteenth International Symposium on Temporal Representation and Reasoning.* 2007. 105-116.

Garofalakis, M. N., R. Rastogi, and K. Shim. "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints." *In Proceedings of the twenty-fifth VLDB Conference.* Edinburg, Scotland, 1999. 223-234.

Garofalakis, M., R. Rastogi, and K. Shim. "Mining Sequential Patterns with Regular Expression Constraints." *IEEE Transactions on Knowledge and Data Engineering*, 2002: 530-552.

Gomez, L., and A. Vaisman. "Efficient Constraint Evaluation in Categorical Sequential Pattern Mining for Trajectory Databases." *In twelfth International Conference on Extending Database Technology.* Saint Petersburg, Russia, 2009. 541-552.

Han, J., ve M. Kamber. *Data Mining Concepts and Techniques.* San Francisco: Morgan Kaufmann Publishers Inc., 2006.

Mierswa, I., Michael. Wurst, Ralf. Kilnkenberg, Martin. Scholz, and Timm. Euler. "YALE: Rapid Prototyping for Complex Data Mining Tasks." *In Proceedings of the twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM Press, 2006. 935-940.

Morzy, T., T. Wojciechowski, and M. Zakrzewicz. "Efficient Constraint-Based Sequential Pattern Mining Using Dataset Filtering Techniques." *In Proceedings of the Baltic Conference.* 2002. 213-224.

Ng, R., J. Han, ve L. Lakshmanan. «Constraint-Based Multidimensional Data Mining.» *IEEE Computer* 32, no. 8 (1999): 226-237.

Pei, J., and J Han. "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth." *In International Conference on Data Engineering.* 2001. 215-226.

Pei, J., J. Han, and W. Wang. "Mining Sequential Patterns with constraints in large databases." *In Proceedings of the eleventh international conference on Information and knowledge management.* Virginia, USA, 2002. 18-25.

Pei, J., ve J. Han. «Constraint frequent pattern mining: a pattern-growth view.» *SIGKDD Explorations* 4, no. 1 (2002): 31-39.

Srikant, R., and R. Agrawal. "Mining Sequential Patterns: Generalizations and Performance Improvements." *In International Conference Extending Database Technology.* Springer, 1996. 3-17.

Witten, H. I., M. Hall, E. Frank, G. Holmes, B. Pfahringer, and P. Reutemann. "The WEKA Data Mining Software." *SIGKDD Explorations* 11, no. 1 (2009): 10-18.

Zaki, M. J. "SPADE: an efficient algorithm for mining frequent sequences." *Machine Learning Journal* 42, no. 1-2 (2001): 31-60.