A Universal Formulation of Sequential Patterns

Mahesh Joshi George Karypis Vipin Kumar
Department of Computer Science
University of Minnesota, Minneapolis
{mjoshi,karypis,kumar}@cs.umn.edu
Technical Report #99-21

May 20, 1999

Abstract

This report outlines a more general formulation of sequential patterns, which unifies the generalized patterns proposed by Srikant and Agarwal [SA96] and episode discovery approach taken by Manilla et al [MTV97]. We show that just by varying the values of timing constraint parameters and counting methods, our formulation can be made identical to either one of these. Furthermore, our approach defines several other counting methods which could be suitable for various applications. The algorithm used to discover these universal sequential patterns is based on a modification of the GSP algorithm proposed in [SA96]. Some of these modifications are made to take care of the newly introduced timing constraints and pattern restrictions, whereas some modifications are made for performance reasons. In the end, we present an application, which illustrates the deficiencies of current approaches that can be overcome by the proposed universal formulation.

1 Introduction

Many scientific and commercial domains have seen an enormous growth of data in recent times. It has become both useful and essential to process this data to learn interesting hidden knowledge from it. The data collected from scientific experiments, or monitoring of physical systems such as telecommunications networks, or from transactions at a supermarket, have inherent sequential nature to them. The discovery of sequential relationships or patterns present in such data is useful for various purposes such as prediction of events or identification of sequential rules that characterize different classes of data.

Different approaches have been taken so far to address this problem. Three prominent approaches are the ones taken in [SA96, MTV97, BWJ96]. Our motivation for studying the universal patterns was obtained when we tried to apply some of these existing approaches to the different kinds of temporal datasets we had to work with. Applying any or only one of these approaches as-is to these datasets would not have helped us discover the kind of information that we were looking for. The reasons for this relate to the two important issues in the process of discovering sequential relationships. One issue is that of the structure of the pattern in terms of its representation and constraints. In this respect, no single existing approach had all the flexibility that we desired. The second issue is the method by which a pattern's strength is computed. Usually, this is based on the number of times a pattern occurs in the given dataset. What matters is the method by which the occurrences are counted. Each of the existing approaches has its own method of counting, mainly motivated by the application domain for which the approach was developed. But, no approach allows the flexibility of supporting multiple methods within a single framework.

We realized that existing approaches can be unified and generalized to a single representation of a pattern, for which a single discovery algorithm can be used. Generality of representation alone is not sufficient, however, in order to make a sequential pattern formulation applicable in multiple scenarios. Different application domains might want to assign strength to a pattern in different manners, because such differences yield different semantics to the discovered patterns. This can be handled by providing multiple ways of counting the occurrences of patterns, such that all counting methods can be implemented with more or less same efficiency, using a single algorithmic framework.

In this paper, we present universal sequential patterns, which unify and extend current approaches to make them more general in terms of representation, constraints, and methods of computing pattern strengths. We begin by specifying the format of the input data that we will work on. Then, we describe the most general form of representing and constraining a sequential pattern in section 3. Different counting methods and their semantic differences are illustrated in section 4. If sequential patterns are used for prediction purposes, we need prediction rules which can be extracted from the patterns. In section 5, we will briefly describe the methods of forming such sequential prediction rules and assigning different measures of interestingness to them. Following this, we give a brief sketch of the algorithm to discover universal sequential patterns in section 6. In the last section, we present some key applications to illustrate the deficiencies of current approaches in certain scenarios that can be overcome by the proposed universal formulation.

2 Nature of Input Dataset

The input is a sequence data characterized by three columns: *object*, *timestamp*, and *events*. Each row records occurrences of events on an object at a particular time. An example is shown in Figure 1. Alternative way to look at the input data is in terms of the timeline representations of all objects (illustrated in Figure 5). It should be noted that the approach taken in [SA96] uses the same format of input data, but approach taken in [MTV97] allows specifying only one object.

Various definitions of *object* and *events* can be used, depending on what kind of sequences one is looking for. For example, in one formulation, object can be a telecommunication switch, and event can be an alarm type occurring on the switch. With this, the sequences discovered will indicate interesting patterns of occurrences of alarm types occurring at a

Object	timestamp	events
A	10	2, 3, 5
A	20	6, 1
A	23	1
B	11	4, 5, 6
B	17	2
B	21	7, 8, 1, 2
B	28	1, 6
D	14	1, 8, 7

Figure 1: Example Input Data

switch. In another formulation, object can be a *day* and event can be a switch or a switch-alarm type pair. This will give interesting sequential relations between different switches or switch-alarm type pairs over a day.

3 Formulation: Representation and Constraints

The sequential relationships among events can be expressed in various forms. The most general form of a valid sequential relationship can be represented by a directed acyclic graph (Figure 2(a)). A node would represent an event or a set of events. Some nodes will be associated with an event-set before the discovery process, which we call *event constraints*. Other nodes would be associated with different possible event-sets during the discovery process. A directed edge from node A to node B would indicate that events of A occur before events of B. A set of numbers is associated with each edge, which we call *edge constraints*. These constraints indicate the allowed separation between the occurrences of events of its incident nodes. For example, each edge can have a 2-element set $\{a,b\}$ associated with it, such that for an edge (A,B) –> (C,D), a represents the minimum required separation between latest event of the (A,B) set and the earliest event of the (C,D) set. On the other hand, b represents the maximum allowed difference in the earliest occurring event among (A,B) and the latest occurring event among (C,D). Each node may also be associated with a set of numbers, called *node constraints*, which governs the definition of when a set of events can be associated with a node; for example, if there is one number $[w]$ associated with a node, then the set of events (C,D) can be associated with that node only if C and D occur within w units of each other. Other than the edge and node constraints, there is one more set of constraints, called *global constraints*, which is imposed on the entire pattern. For example a constraint of the form $< ag,bg >$ imposed on the entire pattern may mean that the total duration of the pattern has lower limit of ag and upper limit of bg. Final characteristic of this dag-based sequential relationship is that each edge can belong to one of the two types: elastic or rigid. An elastic edge can be extended into multiple edges by adding nodes dynamically in succession during the discovery process, as shown in Figure 2(a). Each of the

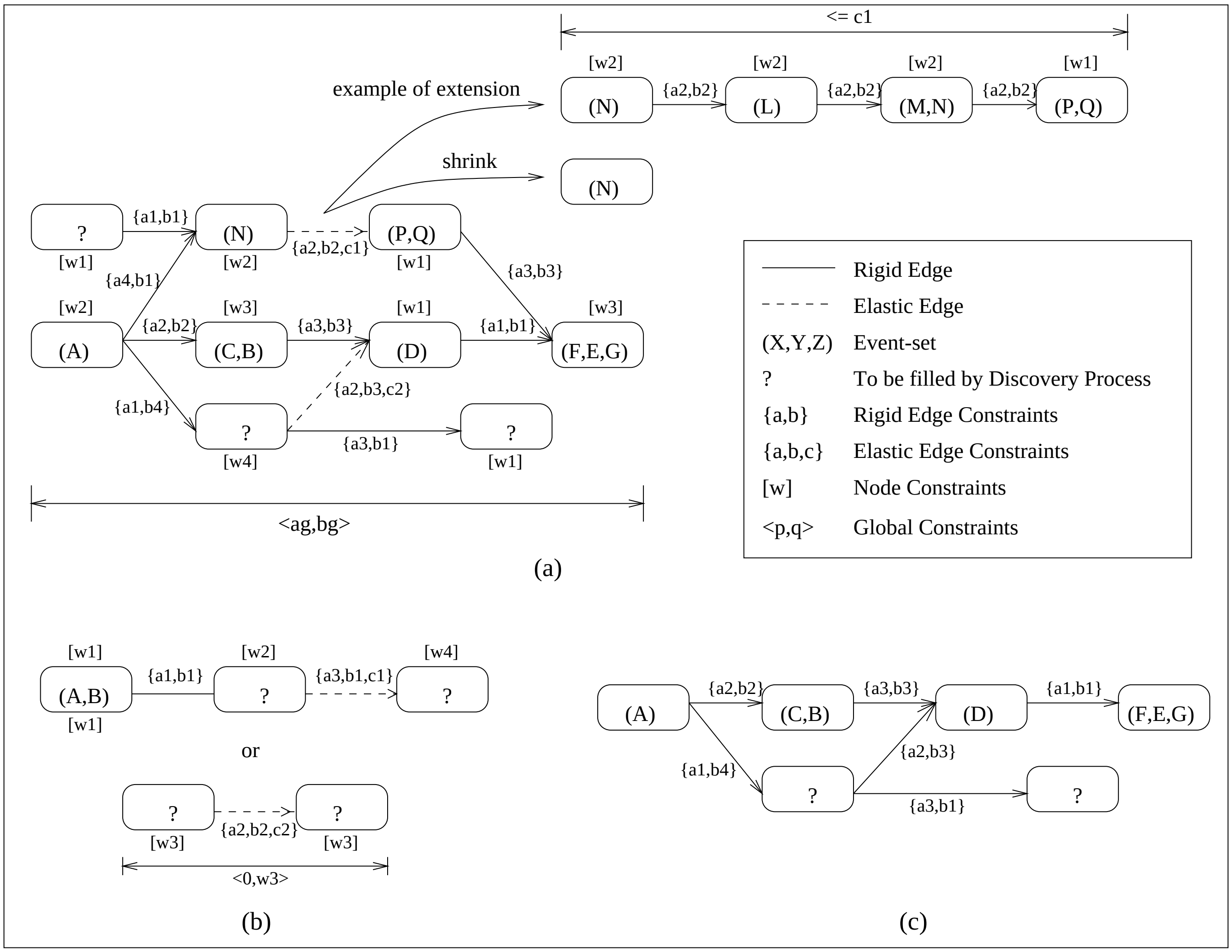

Figure 2: (a) Most general formulation of a sequential relationship, (b) Single path formulation, (c) Formulation due to [BWJ96].

newly added edges would have same edge constraints as the original elastic edge, and each of the newly added nodes would have same node constraints as those of the starting node (A in case of an edge from A to B). The extension capability of an elastic edge is limited by a third constraint on the elastic edge, which limits the entire duration of the extended edge. This is denoted by c in the triplet $\{a,b,c\}$ associated with each elastic edge. An elastic edge can also be shrunk by collapsing one of its incidence nodes onto the other. A rigid edge cannot be changed in this manner during the discovery process.

No work has been done so far on the most general form of the sequential relationship described above. However, simplified versions of this dag have been used in the literature. Two prominent approaches have been taken so far for identifying sequential relationships. One approach, shown in Figure 2(b), restricts the nature of the relationship to a single path of the dag as presented in [SA96] and [MTV97]. In both these works, there is only one elastic edge in the dag, and there are no event constraints. The second approach taken in [BWJ96] assumes the dag to have single root node (no incoming edges) and the structure of the dag is assumed to be fixed during the discovery process; i.e. all the edges are rigid. Figure 2(c) represents this approach. This approach allows event constraints. It also allows different edge constraints to be specified in different time units (or granularities).

Although the dag-based approach described in Figure 2(a) is the most generic in nature, it can be looked at as a compact way of representing multiple single-path sequential relationships. More precisely, a dag can be broken into all its constituent single path relationships by enumerating all the paths between roots and leaves. Root here is a node with no incoming edges, and leaf is a node with no outgoing edges. With this viewpoint, the algorithm which discovers the relationship represented by a dag can be visualized as multiple passes of an algorithm which discovers single path relationships. With this argument, there is no loss of generality if only the relationship represented by a single path of the dag is considered as the universal formulation. However, it should be pointed out that discovering relationships directly in the form of a dag can be more efficient than discovering relationships along all such individual paths, because it has a potential to avoid the repitition of work that would be incurred if individual paths are discovered independently. For the purposes of illustrating the issues of this paper, we assume that the dag is broken into all its constituent paths (or chains). *We refer to such single path relationships with all its event, node, edge, and global constraints, and two types of edges (rigid and elastic) as the univeral sequential pattern.* The node, edge, and global constraints are together refered to as *timing constraints*. It should be noted that all edge constraints are specified in same units (or granularity). Multiple granularities as used in [BWJ96] could be converted to single (finest) granularity before the discovery algorithm.

Before describing the details of universal sequential patterns, it might be worthwhile to note the differences and similarities between the approaches taken in [SA96, MTV97] in the framework of the general form representation that was presented above. As was noted earlier, the approaches are similar in the sense that they both discover relationships along a single path, and they both do not have any event constraints. The formulations, however, differ in two aspects. One is their ability to specify timing constraints, and the other is how they count the occurrences of a candidate pattern in a given dataset. The difference in

Figure 3: Comparing Timing Constraints in sequential pattern formulations of (a) Generalized Sequential Patterns due to [SA96], and (b) Episodes due to [MTV97].

the counting part will be elucidated when we present our unifying formulation in following sections, but the difference in the timing constraints can be made clear by Figure 3. The approach taken in [SA96] has no global constraints, whereas the approach taken in [MTV97] has no edge or node constraints.

3.1 Universal Formulation in Detail

The universal formulation and an example of discovered pattern are given in Figure 4.

A universal sequential pattern is represented as a sequence of the form (A) (C,B) (D) (F,E,G). Here A, B, etc. are the *events* that are *timestamped*. They can be telecommunication alarm types, switch-alarm type pairs, or items bought by a customer in a transaction, or anything that can have a timestamp. Here timestamp is used as a generic term to denote a measure of temporal nature. The example sequence above says that occurrence of event A is followed by occurrence of *event-set* (C,B) which is followed by occurrence of event D, and so on.

The occurrence of particular events can be fixed beforehand (event constraints). For the discovery process to be meaningful, at least one node should have its event-set unspecified. With event constraints present, the discovery algorithm can be optimized to reduce the search space, but in the remainder of discussion, we assume that there are no event constraints.

The node, edge, and global constraints translate into various input parameters which govern the times at which the events in a sequence can occur. Figure 4 illustrates these parameters, called timing constraints.

Here is the description of the parameters:

- ms **: Maximum Span** : The maximum allowed time difference between the latest and earliest occurrences of events in the *entire* sequence.

- ws **: Event-set Window Size** : The maximum allowed time difference between the latest and earliest occurrences of events in any *event-set*.

- xg **: Maximum Gap** : The maximum allowed time difference between the latest occurrence of an event in an event-set and the earliest occurrence of an event in its immediately preceding event-set.

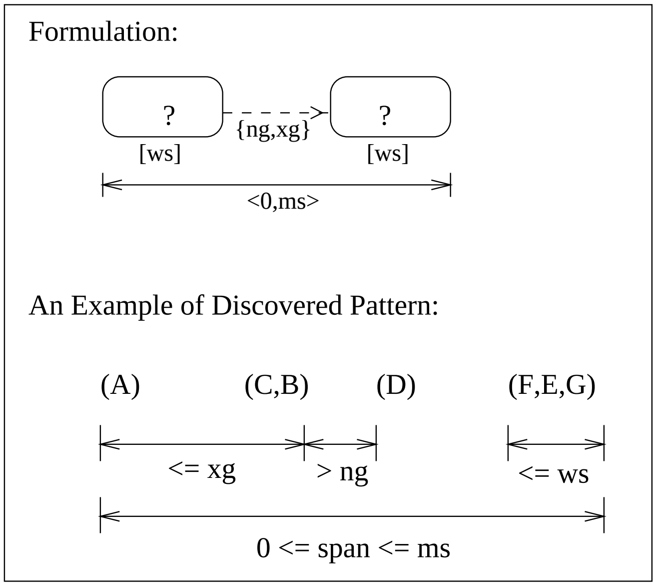

Figure 4: Universal Formulation of Sequential Patterns

- ng **: Minimum Gap** : The minimum required time difference between the earliest occurrence of an event in an event-set and the latest occurrence of an event in its immediately preceding event-set.

It can be seen that as far as timing constraints are concerned, above formulation is equivalent to the formulation in [SA96] if $ms \rightarrow \inf$; and it is equivalent to the episode formulations in [MTV97]: serial episodes if $ws < 0$, $xg \geq ms$, and $ng = 0$, and parallel episodes if $ws = ms$, $xg \geq ms$, $ng \geq ms$. Actually, for the formulation to be exactly equivalent to those in [SA96] or [MTV97], the choice of counting method also matters, which is discussed in the next section.

4 What is an interesting sequential pattern?

A sequence is said to be *interesting* if it occurs *enough* number of times satisfying the given timing constraints (ms, ws, xg, ng).

There are two issues here. First issue is, how the sequence occurrences are counted. This deals with different counting methods, and we will shortly elaborate on it in great detail. The second issue is, how many occurrences are *enough*? This is determined by the *support threshold*, which is an input parameter. After doing the counting, the sequences which do not occur *enough* number of times are filtered out. The support threshold can be specified in terms of absolute count, or percentage with respect to some basis. The support threshold is commonly used as a measure of interestingness of a pattern because of one of its very important properties. The property is, a subsequence of any sequence has at least as much support as the sequence, or put another way, the support for a sequence cannot be any greater than the support of any of its subsequences. This property helps to reduce the

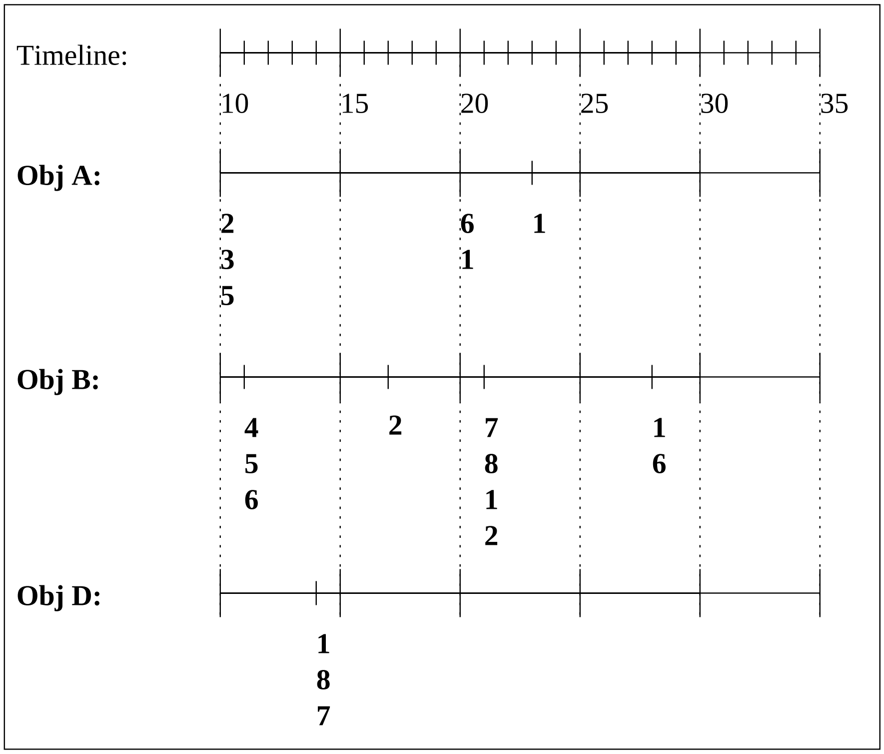

Figure 5: Time-line representation of input data of Figure 1

algorithmic complexity of discovering interesting patterns by allowing a systematic growth of patterns based on the apriori principle [SA96].

Coming back to the issue of *how-to-count*, there are five different ways defined for counting the number of occurrences. These can be divided into three conceptual groups.

First group, which consists of COBJ, just looks for an occurrence of a given sequence in an object's timeline. One occurrence is enough to ensure that the sequence occurs in that object. Second group is based on counting the windows in which the given sequence occurs. This consists of CWIN and CMINWIN. Third group is based on counting the distinct occurrences of a sequence. This consists of CDIST and CDIST_O. Although the occurrences are restricted to be contained in a window of size ms (maximum span), this group is different from the window-based counting group, because occurrences describe a cause-effect relationship directly based on the events themselves. The window-based group takes a different approach. It is based on the premise that the user is observing events occurring in a window, and is looking for the relationship to be exhibited in the window at least once.

This difference will become clearer as we describe the methods below. Figure 5 is used to illustrate their definition, whereas Figure 6 will elucidate the difference between different counting groups. Which approach is suitable will depend on the specific application that the user has in mind, and on user's domain expertise in the area. The applications in section 7 will throw more light on this aspect.

- (counting method = **COBJ**) One occurrence per object.
  The count here indicates the number of objects in which the sequence appears. For example, sequence (2) (1,6) with ms=20, ws=0 has two occurrences: for A, (2) at t=10 and (1,6) at t=20, and for B, (2) at t=17 and (1,6) at t=28. Note that, although (2) (1,6) appears in B one more time with (2) at t=21 and (1,6) at t=28, it is counted only once.

- (counting method = **CWIN**) One occurrence per span-window.
  Span-window is defined as a window of duration equal to span (ms). Consecutive span-windows have one time unit's difference in their respective start and end times. They move across the entire time duration of each object, but none of the span-windows spans across two different objects. The counts for all the objects are added up. In Fig. 3, with ms=20 and ws=5, sequence (2) (1,6) has 23 occurrences: it appears in 10 windows for A, with windows from t=1 to t=10, and in 13 windows for B, with windows from t=9 to t=21. Note that, the window of span 20 starting at t=9 is defined as the [9,29) interval.

- (counting method = **CMINWIN**) Number of Minimal Windows of Occurrence.
  A minimal window of occurrence is the smallest window in which the sequence occurs given the timing constraints. In other words, a minimal window is the time interval such that the sequence occurs in that time interval, but it does not occur in any of the proper subintervals of it. This definition can be considered as a restrictive version of CWIN, because its effect is to shrink and collapse some of the windows that are counted by CWIN.

  This method is similar in spirit to the concept of minimal occurrences in [MTV97], but there is one difference. In [MTV97], there is no duration limit on the minimal occurrence. Whereas, in our CMINWIN method, the size of the minimal window is limited by the maximum span (ms) constraint.

  All the minimal windows of occurrences are counted for each object, and then they are added up over all objects. For example, with ms=20, sequence (2) (1,6) has only two minimal window occurrences, one for A and one for B. The occurrence in B with (2) at t=16 and (1,6) at t=28 is not a minimal window occurrence because it contain another smaller window of occurrence with (2) at t=21 and (1,6) at t=28, which indeed is a minimal window of occurrence. It can be seen that many windows that were counted by CWIN method are collapsed into these two windows. For example, all those windows in object A that had the pattern same pattern (2)(1,6), with (2) at t=10 and (1,6) occurring in t=[20,23], are shrunk and collapsed into one minimal window [10,20].

- (counting method = **CDIST_O**) Distinct Occurrences with Possibility of Event-Timestamp Overlap.

  An distinct occurrence of a sequence is defined to be the set of event-timestamp pairs that satisfy the specified timing constraints, such that there has to be at least one new event-timestamp pair different from the previously counted occurrence. Counting all such distinct occurrences results in CDIST_O method.

The number of occurrences counted using this method depends on the direction in which an object's timeline is scanned. We assumed that the timeline is scanned in the direction of increasing timestamps.

As with all previous methods, the sequence occurrence must have all its events happening on the same object. All occurrences of the sequence are added over all objects.

As an example, with ms=20, sequence (2) (1,6) has three distinct occurrences when overlap is allowed, one for A and two for B. The occurrence in B with (2) at t=21 and (1,6) at t=28 is a distinct occurrence because (2) at t=21 is the new event-timestamp pair from the previously counted (2) at t=17 and (1,6) at t=28 occurrence.

- (counting method = **CDIST**) Distinct Occurrences with No Event-Timestamp Overlap Allowed.

  In CDIST_O above, two occurrences of a sequence were allowed to have overlapping event-timestamp pairs. In this CDIST method, we don't allow any such overlap. So, effectively when an event-timestamp pair is considered for counting some occurrence of a sequence, it is flagged off and is never again considered for counting occurrences of that particular sequence for that particular object.

  As in CDIST_O, the sequence occurrence must have all its events happening on the same object, all occurrences of the sequence are added over all objects, and the timeline is scanned in forward direction.

  As an example, with ms=20, sequence (2) (1,6) has only two distinct occurrences, one for A and one for B. The occurrence in B with (2) at t=21 and (1,6) at t=28 is not a distinct one because events 1 and 6 at t=28 are already used in counting the first occurrence which has (2) at t=17.

The relationships and differences between methods CWIN, CDIST, CDIST_O, and CMINWIN are further clarified by Figure 6, which assumes presence of only one object. If n_O indicates the number of occurrences counted with method O, then it can be noted that $n_{CMINWIN} < n_{CWIN}$ and $n_{CDIST} < n_{CDIST\_O}$. Look at relationships across the columns of Figure 6(a). It can be noted that CWIN and CDIST_O are similar in spirit because they count *all* the windows and occurrences, respectively, and CMINWIN and CDIST are similar because they count the *minimal* windows or occurrences. The cause-effect philosophy followed by occurrence based approaches and the observation-window philosophy taken by window based approaches clearly yield different occurrence counts as illustrated in part (b) of Figure 6.

With this set of different counting methods, it is interesting to note that when COBJ method is used with $ms \rightarrow$ inf, the formulation is exactly equivalent to that in [SA96]. The formulation is exactly equivalent to that in [MTV97], when CWIN method is used along with appropriate values of (ms,ws,xg,ng) as indicated in the section 3. Of course, our formulation allows more than one objects, whereas the formulation of [MTV97] does not allow it, and our formulation discovers the hybrid episodes (containing both serial and parallel) with equal ease as that of serial or parallel episodes.

(a)

	Count All	Count Minimal
Count Windows	CWIN	CMINWIN
Count Occurrences	CDIST_O	CDIST

(b)

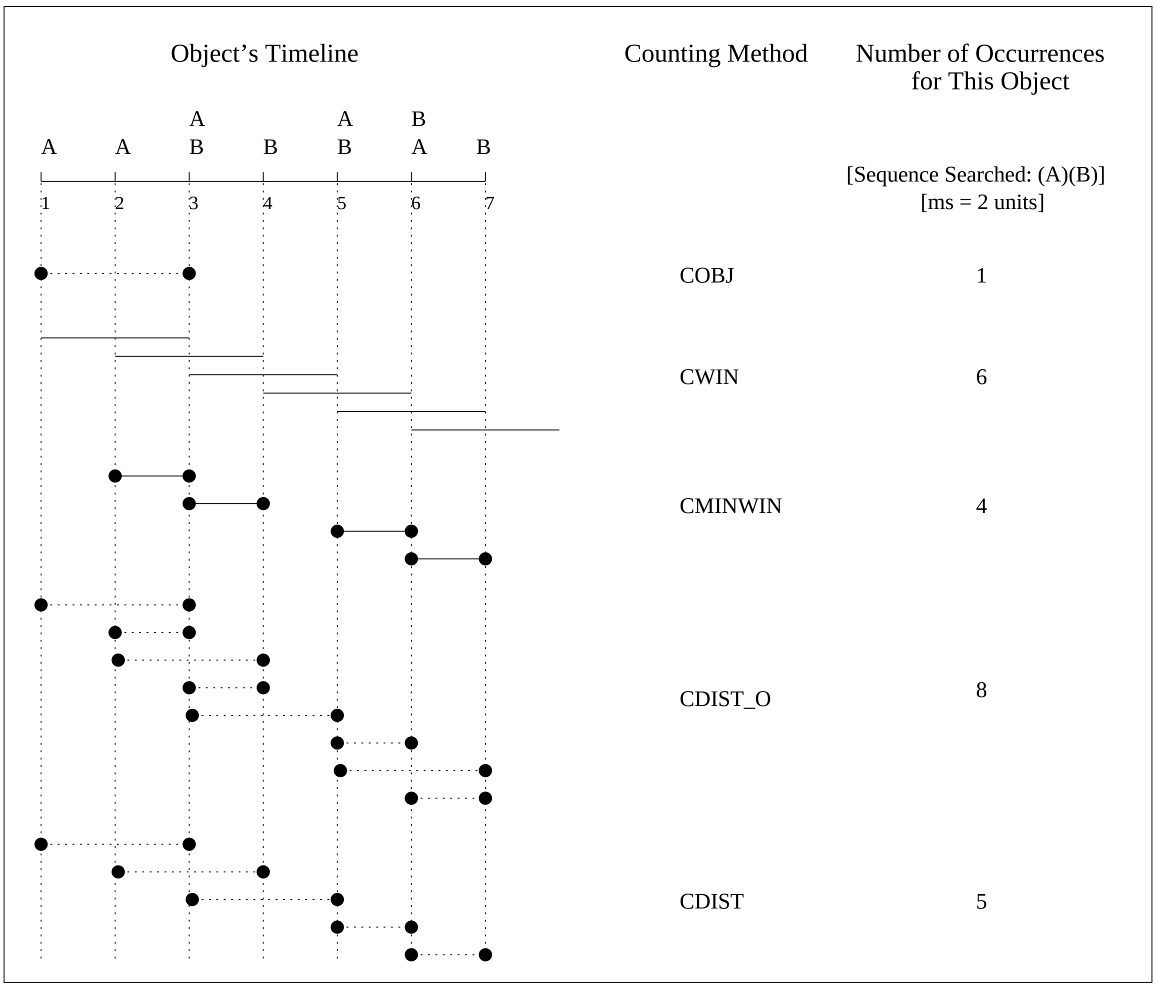

Figure 6: Comparing different counting methods. (a) Relationship among methods that count multiple occurrences per object (all but COBJ), (b) Differences among methods.

If the percentage based support threshold is used, then we need to determine the basis for this support percentage. It depends on the counting method above is used. For the method COBJ, the basis is total number of objects in the input data. For methods CWIN and CMINWIN, the basis is the sum of the total number of span-windows possible in all objects. For methods CDIST and CDIST_O, the basis is maximum number of *all possible* distinct occurrences of a sequence over all objects, which is the number of distinct timestamps present in the input data of each object.

5 What is an interesting sequential rule ?

Section 4 discussed interesting sequential patterns. For each interesting sequential pattern discovered, a sequential rule predicting the occurrence of last event-set in the sequence can be deduced. For example, for a pattern (A) (B C), the rule deduced is of the form, If A occurs, then event-set (B,C) occurs within the timing constraints specified. This rule is represented as (A) $\rightarrow$ (B,C). Similarly, for a pattern (A,B) (C) (D), the rule predicting occurrence of D after occurrence of (A,B) (C), is formed as (A,B) (C) $\rightarrow$ (D).

Stating it formally, rule S1 $\rightarrow$ IS, predicting IS, is formed using the sequential pattern, S : S1 (IS), where S1 is the subsequence formed by omitting the last event-set, IS. We say that this rule is *interesting* or *strong* if occurrence of S1 predicts occurrence of IS with a large *significance*. Significance is defined as Confidence / (co(IS) / Support basis), where Confidence is in turn defined as co(S) / co(S1). Here, co(S), co(S1) and co(IS) denote the number of occurrences of sequential patterns S, S1, and IS, respectively. Obviously, the same method must be used for counting occurrences of S, S1, and IS. Confidence is a number less than 1, and significance can be any number greater than 0.

Intuitively, confidence tells the conditional probability with which one can predict the occurrence of the consequent, IS, after seeing an occurrence of the antecedent sequence, S1. A rule which occurs with high confidence and relatively less occurrences of the consequent, together implying a high significance, is *strong* because the antecedent predicts *most* of the consequent's occurrences with high confidence. There is another characteristic of the rule called *coverage* that tells us about the fraction of times the entire sequential pattern occurs with respect to the number of times the consequent occurs. Coverage is defined as co(S) / co(IS). A rule with high significance and high coverage has a better predictive power.

When the sequential pattern has only one event-set in it, the rule formed will predict the occurrence of these events happening within the *ws* duration. In this case, the confidence, significance, and coverage numbers associated with the rule are formed by averaging the corresponding numbers over all the rules that predict occurrence of each of the events in the event-set. As an example, if the pattern is (A,B,C), then three rules will be formed (A,B) $\rightarrow$ (C), (B,C) $\rightarrow$ (A), and (A,C) $\rightarrow$ B; the significance, confidence, and coverage are computed for each of these three rules and then they are averaged out.

Once the measures of interestingness are computed for all the discovered rules, they can be ordered in any way the user wishes. Usually, an ordering that lists the rules in decreasing order of significance will list interesting and strong rules at the top. Ties can be broken using confidence, coverage, and support in that order.

Structure of the Algorithm: (Based on GSP [SA96])

```
form Set of Large Sequences, L_1 each having 1 event;
k = 2;
while ( L_{k-1} is not empty )
    join L_{k-1} with L_{k-1} to form C_k, set of Candidates having
    k events; [Store C_k in a hash tree access structure] ⟵

    for each object's timeline
        repeat
            traverse hash tree to search for presence of possible candidates; ⟵
            count the occurrences of candidates at leaf nodes;                ⟵
        until (no more traversing possible)
    end

    form L_k from C_k by retaining only the large candidates (having count above
    support threshold);
end
```

Figure 7: Structure of the Algorithm. Similar to GSP [SA96]. The phases where modification is made are shown with an arrow.

6 Discovery of Universal Sequential Patterns

In this section, we briefly describe the algorithm. The overall structure of the algorithm is same as that of GSP [SA96], as outlined in Figure 7. The modifications are made to the join phase and the counting phase.

If there are no event costraints, then the join phase is identical to that of GSP. In case of event constraints, join phase can be modified to produce only those patterns which satisfy the event constraints.

The counting phase is modified at two places. One is in the hash tree traversal part. Other is in the part which counts the occurrences of sequences appearing at a leaf of the hash tree.

In the hash tree traversal, an object's timeline is streamed through the hash tree such that all events in the timeline are considered valid at the root node. Once an occurrence of an event is fixed at a node, only those other event occurrences which fit within the timing constraints (ms,ng,xg,ws), are considered eligible for hashing at that node to reach a child node. The traversal continues until a leaf is reached. Two modifications are made in this traversal algorithm. One is to take into account the maximum span, ms, parameter, which is absent in GSP. It is accounted for while searching for the eligible event occurrence at the child node. In GSP, the occurrence is searched for in the range [t-ws, t+max(ws,xg)]. In modified algorithm the range is [t-ws, max(t+max(ws,xg), t_0+ms)], where t is the time at which an event is found at the parent node and t_0 is the earliest occurrence time of the events found so far in the path taken from root to the parent node. Another modification in hash tree traversal is made for performance reasons. The modification is to identify the paths in the tree which will or should never be visited again by the given object's timeline and to flag such paths in order to avoid further unnecessary and time-consuming traversals along them. This is especially very important for efficiency purposes when a timeline is very long and has large number of events in it.

The other part where the GSP algorithm is modified is when a leaf node of a hash tree is reached. The counting algorithm given in GSP stops after finding the first occurrence of a sequence in the timeline. This is okay for the COBJ counting method, but for other counting methods, the search must continue beyond the first occurrence until the entire timeline is consumed. Also, the event-timestamp pairs need flagging especially when CDIST method is used. The modification uses the GSP's counting algorithm in its inner loop, and the outer loop is executed to find next occurrences according the counting method chosen by the user.

7 Applications of Universal Sequential Patterns

Let us see one representative application, where universal formulation presented in this paper is required and the formulations suggested so far in the literature are not suitable.

- **An Illustrative Application**: Discovering Consumer Buying Patterns.

  Let us consider an example of a grocery store.

  Goal: To find out "which items' sales trigger sales in other items within a period of one week" so that the information can help in managing weekly inventory.

  Formulation:
  Object: Customer
  Event: Items bought by the customer
  Timestamp: Date of transaction

  Constraints:
  $xg = 2$ days, $ng = 0$ days, $ws = 0$ days, $ms = 7$ days.

  This constraints restrict all the items in an event-set to be bought on the same day (ws=0), the pattern span does not exceed 7 days (ms), and a pattern like (Eggs)(Bread) will be supported by a customer only if he buys Eggs today and Bread tomorrow or day-after-tomorrow (xg=2). Finally, ng=0 means that items A and B bought on the same day cannot occur on two different nodes separated by an edge.

  Counting Method:

Let us consider a pattern (Orange Juice)(Mayonnaise)(Frozen Burger).

COBJ: If pattern above is found to be frequent with this method, then it can be concluded that *many customers* who buy Orange Juice today will buy Mayonnaise within a couple days followed by Frozen Burger within a couple of days more, *but* they will buy all these items within a week. The frequent patterns generated with this method will tell *how many customers* are likely to exhibit that pattern. Note that, this method does not try to increase the strength of the pattern if some of the customers exhibited this pattern multiple number of times, which might be important in some cases.

CDI ST: If the pattern above is found to be frequent with this method, then it implies that many distinct occurrences of the pattern were exhibited when they were added up over all customers. If the goal is to increase the sales of Frozen Burgers, then this counting method gives more practical meaning to pattern. This is because distinct occurrences with no overlap mean that we could find sufficiently many distinct sequences of these items without counting any item more than once. So, the count reflects on how many cans of orange juice and how many Mayonnaise and Burger packs were actually sold.

CWIN, CMINWIN, CDIST_O: Although these counting methods can be applied, they may not be valuable from the practical viewpoint, because none of them would reflect on the true sales of the items involved in the pattern because overlap is allowed in all of them. This means that same can of Orange Juice can contribute to many occurrences or many windows.

This example illustrates that not every counting method can be applicable in all the domains. Nature of the application domain and user's knowledge regarding the domain will determine which counting method to use.

Where Does Universal Formulation Help?:

It seems that the generalized sequential patterns formulation of [SA96] could have helped in this scenario, but that formulation has only one counting method (COBJ), which does not allow to gain more insight into the patterns which are given by the other method (CDIST) of universal formulation. Moreover, generalized sequential patterns of [SA96] do not place any limit on the total pattern duration. Hence, if the available data is collected over a large number of weeks, then in order to make generalized sequential patterns applicable, multiple 7-day-long datasets would need to be formed and analyzed separately. Many issues would arise in that case, such as which 7-day intervals to choose, how to combine the patterns generated with each dataset, etc.

Also the method of [MTV97] is not applicable. The fact that the method does not allow multiple objects needs to be handled first. This requires extra preprocessing of input dataset, which includes merging of the datasets for all the customers into a single dataset and separating individual customer's timelines by a gap greater than ms value, so that patterns do not span across two different customers. Even if this extra cost of preprocessing is tolerated, the method does not support COBJ or CDIST counting methods. The only counting method it supports is CWIN, which does not seem to have much practical value in this scenario (as explained above). Also, it does not allow to specify the xg and ng constraints, which can be crucial in this application.

Many more applications are possible. We will list a few of them here.

- **Possible Application Areas for Universal Sequential Patterns:**

  1. Discovering sequential relationships between different telecommunication switches and alarms triggering on them.
  2. Analyzing data from scientific experiments conducted over a period of time.
  3. Discovering relationships between stock market events (e.g. fluctuations happening on market indices, individual stock prices).
  4. Analyzing medical records of patients for temporal patterns between diagnosis, treatment, symptoms, and examination results, etc.
  5. Discovering Patterns Among Different Socio-Economic Events.

It should be noted that each application domain will require a careful combination of timing constraints, event constraints, and counting methods, in order to produce meaningful patterns. With universal formulation, one can easily try out multiple such combinations, specify different structures of the relationships, and the same algorithm is applicable in all the cases. This is real strength of the universal sequential patterns. The models that were previously suggested in the literature do not have enough set of constraints for a user to focus on specific kinds of patterns, and they do not support different counting methods. Allowing different counting methods to be supported in one unifying formulation, ultimately allows a user to encode different hypotheses he/she has about the sequential relationships. This can range from hypothesizing nothing but a few timing constraints to putting all possible constraints encoding all the previously known knowledge the user has about the application domain.

References

[SA96] R. Srikant and R. Agrawal, *Mining Sequential Patterns: Generalization and Performance Improvements*, Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT), Avignon, France, March 1996.

[MTV97] H. Mannila, H. Toivonen, and A. I. Verkamo, *Discovery of frequent episodes in event sequences*, Technical Report C-1997-15, Dept. of Computer Science, University of Helsinki, 1997.

[BWJ96] C. Bettini, X. S. Wang, and S. Jajodia, *Testing Complex Temporal Relationships Involving Multiple Granularities and Its Application to Data Mining*, Proc. of ACM PODS'96, pp.68-78, Montreal, 1996.