Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference

Stretchy Time Pattern Mining: A Deeper Analysis of Environment Sensor Data

Carlos Roberto Silveira Junior, Marcela Xavier Ribeiro, Marilde Terezinha Prado Santos
Department of Computing, Federal University of São Carlos
Washington Luís, km 235 – SP–310, 13565-905
São Carlos, São Paulo, Brazil

Abstract

Mining sequential patterns on environment sensor data is a challenging task; the data can present noises and may also contain sparse patterns, which are difficult to be detected. The knowledge extracted from environment sensor data can be used to determine climate changes. However, there is a lack of methods that can handle this kind of database. In this paper, we propose a method to mine sequential patterns in sparse, incomplete and noisy sensor data. The proposed method, called Stretchy Time Windows (STW), allows the mining of sequential patterns that present time gaps between their events. We propose an algorithm to implement STW, called Miner of Stretchy Time Sequences (MSTS). The proposed algorithm works with sequences of any size and uses a balanced strategy to analyze the search space. Our experiments show that MSTS returns sequences that have a longer period of analysis than GSP a traditional frequent pattern mining algorithm. In fact, 5 times larger than GSP and higher number of patterns (2.3 times) when compared to previous methods.

Introduction

Mining patterns on environment sensor data is a complex task because the data present a set of characteristics that makes the analysis hard. The data are sorted by the space-time order, the data gathering moment and site. The database can either be incomplete or present noises, because of miscalibrated measuring devices, and missing data. Hence, sensor data are incremental and irregular. Normally, during data collection time, many ad hoc procedures are used (Barseghian 2010); the data are stored without standard. Furthermore, there are constantly adding of new data in the database.

The knowledge extracted from the sensor data analysis can be used, for example, to determine the best time to sow a particular crop thus maximizing profits, it can improve weather forecasts (based on data behavior). At the moment, there is a lack of methods that effective extract patterns from environmental sensor data. Sequential pattern mining is used on databases whose data are sorted by the time order of event occurrences. It is related to the task of finding sequences of events that frequently occurs in a database, keeping the order of events between their occurrences. That kind of task was firstly proposed by (Agrawal and Srikant 1995).

The task of finding sequential patterns is also difficult because sequential patterns should keep the time order between event occurrences. When working with natural environment sensors, the data can present noise and error because the measurement devices are not always as calibrated as they should be. Also, the task of mining sequential patterns gets even more difficult in this domain because the natural phenomena usually take time to show their effects.

In fact, the size of relevant gaps (time intervals) between events in a sequence can be defined by domain experts, allowing the mining of patterns that best represent the knowledge hidden in the database. To solve the problem of mining patterns that have time gaps between the sequence events, we propose a new algorithm called *Miner of Stretchy Time Sequences* (MSTS), which uses a new technique of windowing to process the sequential data. MSTS is explained in Section and it is employed to allow time gaps to occur between events (items) in a sequential pattern. The size of the duration of the maximum time gap is set by the user. Also, the proposed algorithm works with sequences of any size, not imposing a size limit to the analyzed sequences, as the previous techniques do.

This paper is organized as follows: Section presents important concepts for this work and related works. Section presents the MSTS Algorithm, Section presents the experiments. Section finishes the paper presenting the conclusions and the future works.

Background and Related Work

A sequential pattern is a sequence of events (itemsets) [1] that keep an order among their occurrences. Let's consider a sequence $s =< i_1 \dots i_n >$, where i_k is an itemset (a nonempty set of items that happen together), $n \geq 2$ and i_{k-1} precedes i_k for $1 < k \leq n$. s' is a sub-sequence of s ($s \prec s'$), if and only if, for all itemset $i_k \in s$, s' has an itemset i' which is sub-itemset of i_k or an empty set keeping

[1] The word "events" is sometimes used instead of "itemsets" because "event" is more semantic to explanations focusing in the application domain. However, "itemset" is ordinarily used to conceptual explanations.

the sequential order of s. The traditional sequential mining algorithms usually separate the sequences in frequent and non-frequent ones. The frequency of a sequence is called support and can be calculated by the Formula $support(s) = \frac{|number\ of\ occurrence\ of\ s|}{|number\ of\ sequences\ on\ the\ database|}$. Support is a value between zero and one ($[0; 1]$) (Huang 2009).

If $support(s)$ is equal or greater than a minimum value of support, usually set by the user, then the sequence s is considered frequent. If it is not, the sequence is considered non-frequent. That is important because the anti-monotonic property says that a frequent pattern is always composed by frequent patterns. Therefore, if a non-frequent pattern is created, it is discarded because it cannot be part of a frequent one (Han, Pei, and Yan 2005).

Based on AprioriAll [2] (Agrawal and Srikant 1995) the algorithm Generalized Sequential Patterns (GSP) was proposed by (Srikant and Agrawal 1996). GSP and AprioriAll are important in this work because MSTS is based on their strategy (generation–and–test of candidates).

GSP receives a vertical database and the value of minimum support as input. In a vertical database (the most common format employed to organize a database), all the tuples are organized by the time of event occurrence. Hence, by using a vertical database, GSP avoids rearranging the data to horizontal mode.

Others algorithms as PrefixSpan (Pei 2001) uses the horizontal mode. In horizontal database, all tuples store an identification (e.g. site of collection of samples) followed by a list of events ordered by time of occurence at the sample collection site. PrefixSpan presents better performance than GSP. However, PrefixSpan usually needs to rearrange the data, which takes time and sometimes it needs the supervision of a domain expert to avoid the insertion of errors in the data. For this reason, we decided to use GSP inside of PrefixSpan to orientate MSTS.

The mining of sequential patterns with time constraint considers some restriction of time domain to search for consistent patterns (Chen 2010). For instance, the time constraint can restrict patterns to occur after a given data and time. That is very used to stream mining because, when working with this kind of data, the old data cannot represent correctly the behavior of the new data. Time constraint can direct the focus of the data mining to a specific period of time (Masseglia, Poncelet, and Teisseire 2009).

In (Nunthanid, Niennattrakul, and Ratanamahatana 2011) the algorithm Variable Length Motif Discovery (VLMD) is presented to extract temporal patterns from sequences. The distance calculated between items is Euclidean. The approach uses a kind of adaptive windowing where overlaps are created without pre–configuration. However, this approach does not allow the mining of time sparse patterns, employing a window search on the new and/or most important data. In the direction, MSTS is projected to mine time sparse patterns and no different level of importance is given to any set of data, being more appropriate to capture the evolution of natural phenomena taken by environment data sensors.

[2]First algorithm to mine sequential patterns.

Another important work is presented in (Zabihi 2010), which proposes the Constrained Fuzzy Sequential Pattern Mining Algorithm. The algorithm extracts Fuzzy Sequences (sequences with numerical values) making unnecessary preprocessing to discretize numerical values. A method, called Fuzzy Sliding Windows, is presented in the paper, this method restricts a time window and allows the candidate generation in it.

The approaches discussed are not effective when working with time sparse (time gaps) and incomplete data, as the data produced by natural environment sensors. To amend this restriction from previous methods, MSTS has the advantage of working with multi-resolution time window when searching for sequential patterns.

The Proposed Algorithm: *Miner of Stretchy Time Sequences*

The *Miner of Stretchy Time Sequences* (MSTS) Algorithm is a new algorithm to find sequential patterns with time constraint. The same time constraint is applied to every occurrence of each pattern. The algorithm finds patterns that present time gaps (time interval between consecutive events). MSTS has an input parameter that specifies how long the time gaps can be (μ). It makes the algorithm adequate to find sequential patterns on environment sensor data because the consequences of an event can take long time periods to be detected. For example, the growth of vegetation takes five days to be perceived after a rainy day.

In our approach, the sequence $n-sequence$, presented by (A), is characterized by a list of the events (itemsets $i_{1,2,\ldots n}$) that could present time gaps ($\Delta t_{1,\ldots n-1}$) where $n \geq 2$ is the number of itemset. If the domain has granularity in days (periods between occurrence of the events are counted in days) the Δt_k is the quantity of days that can take between i_k and i_{k+1} where $1 \leq k < n$. Each sequence must obey the constraint $(\sum_{k=1}^{n-1} \Delta t_k) \leq \mu$; the time gaps presented by each occurrence of every pattern should not be greater than μ. A n-sequence takes at least n time units (days, weeks...) to occur (event after event). In our proposal the same n-sequence can take at most $n + \mu$ time units to occurs completely.

$$\boxed{n - sequence =< i_1\ \Delta t_1\ i_2\ \ldots\ \Delta t_{n-1}\ i_n >} \quad (A)$$

MSTS, presented in Algorithm 1, receives as input a vertical database db and the minimum support value $minSup$ –as GSP; MSTS also receives the maximum time gap value μ. The output is the set of frequent sequential patterns F. The first step of MSTS consists in finding in the database db the set of frequent itemset (line 1) and create the 1-sequences (line 2) using the frequent itemsets. After that step, MSTS checks if the support of all combination between each 1-sequence (pattern p) with all frequent itemset (itemset ι) is greater or equal to $minSup$ (line 6). The combination between the 1-sequences and the frequent itemsets is done concatenating the itemsets to the sequences (creating the 2-sequences candidate [3]). If a 2-sequence is frequent then it

[3]Sequences candidate are those whose support values were not checked yet.

Algorithm 1: The *Miner of Stretchy Time Sequences* algorithm (MSTS).

```
   Input: vertical database db, minSup, μ
   Output: set of patterns F
   /* Finding frequent itemsets and generating 1-sequences.   */
1  C ← {frequent itemset of db} ;
2  F ← C ;
   /* Generating and testing sequences.                       */
3  foreach pattern p ∈ F do
4      find ← false ;
5      foreach itemset ι ∈ C do
           /* Checking support of sequence candidate.         */
6          if checkingOccurrence(p,ι,μ) / |number of sequences in bd| ≥ minSup then
               /* If the sequence candidate is frequent then
                  adding in F                                  */
7              add(concat(p, ι), F) ;
8              find ← true ;
9      if find then
           /* Removing subpattern.                            */
10         remove(p, F) ;
```

Algorithm 2: Algorithm of the it Stretchy Time Windows (STW) method.

```
   Input: pattern p, itemset ι, μ
   Output: count /* Number of occurrence of pattern p        */
1  function checkingOccurrence
2  count ← 0 ;
3  p' ← concat(p, ι) ;
4  foreach occurrence occ of pattern p do
       /* Calculating the number of tuples where ι can be found
          to consider the occurrence.                        */
5      numberOfTimeGaps ← (timeOfLastElement(occ) −
       timeOfFirstElement(occ) + 1) − sizeOf(p) ;
6      window ←
       MIN(μ − numberOfTimeGaps, Δ(nextOcurrence(p))) ;
       /* Searching for ι in calculated tuples.              */
7      for k ← 1 to window do
8          if happen(ι, tupleAtTime(timeOfLastElement(occ) + k))
           then
               /* ι found.                                    */
9              count ← count + 1 ;
10             break ;
```

is added to the set of frequent patterns F (line 7); making the set bigger. The next step is to remove the sub-pattern. If the 1-sequence (pattern p) has generated a bigger pattern then this sequence is a sub-pattern and it should be removed from frequent pattern set F. When all 1-sequences are processed, MSTS processes the 2-sequences (that are in set F) using the same process (line 3). In this way, all combination of each 2-sequence (now represented by pattern p) with all frequent itemset (ι) has its frequency checked (line 6), the frequent 3-sequences are added in F (line 7) and the sub-patterns are removed (line 9 and 10). That loop still running until the combination does not generate any frequent pattern anymore.

Although MSTS is based on algorithms that apply generation-and-test of candidate strategy (e.g. AprioriAll and GSP), it presents a huge difference on the way algorithms count the pattern occurrences. Our proposed method of counting occurrences, *Stretchy Time Windows* (STW), in contrast to GSP and PrefixSpan, aims, by configuring the parameter μ, to check a no fixed number of tuples looking for the occurrence of the pattern while GSP and PrefixSpan check in a fixed number of tuples the occurrence of the pattern.

The STW method, presented in Algorithm 2, uses the parameter μ, which says how long the maximum time gap can be. The STW method works in the following way: every time a pattern $p' =< i_1 \dots i_n\ i_{n+1} >$ is generated based on an old frequent pattern $p =< i_1 \dots i_n >$ (line 3), the existence of p' is checked (line 4 until 10). Note that, $p' \succ p$ (p is part of p'). The way of checking is: for all occurrences of p, the algorithm sees how sparse that occurrence is (line 5) and calculates a number of tuples that can be checked to find the new item, $window$ (line 6). $window$ should be the minimum value between $\mu - numberOfTimeGaps$ (μ decremented the existent time gaps) and $\Delta(nextOcurrence(p))$ (distance between the last itemset of the analyzed occurrence of pattern p and the last itemset of next occurrence of p) because the Time Search Window cannot be big enough to cover the next occurrences of pattern p (line 6). The algorithm keeps a vector with all timestamps where p has occurred and a vector to store all frequent itemsets. Hence, to find the p' counter value, it is necessary to calculate how many times i_{n+1} (frequent itemset concatenated in p to generate p') happened after p (line 7 to 10).

The *Stretchy Time Windows* (STW) is an alternative to handle with noisy and missing data. The errors can be solved because the errors can be considered as a time gap. In fact, an error or a missing data is considered as a moment where nothing happens: a time gap. Consider the hypothetical situation where there are two non–frequent patterns $s_1 =< a\ b\ c >$ and $s_2 =< a\ d\ c >$. The pattern $s =< a\ c >$ is frequent although its events do not happen one immediately after one. MSTS finds that $s =< a\ c >$ if it is frequent and if the value of the maximum search window (μ) can cover c's occurrences enough times to consider s frequent. That way MSTS can handle databases that present noises and data inconsistency.

Examples of *Stretchy Time Windows* Process

Consider the three situation present in Figure 1. To the Situation 1, Figure 1(1), consider the 1-sequence $p =< (A\ B) >$ (that happens once) and the goal is to count how many times $p' =< (A\ B)\ G >$ happens. Considering $\mu = 5$ time unit. The pattern p does not present time gaps, so $numberOfTimeGaps = 0$. There is no other occurrence of pattern p than $\Delta(nextOccurence(p)) = 4$ (distance for the end of the database). It makes $window = 4$ ($\mu - numberOfTimeGaps = 5$ and $\Delta(nextOccurence(p)) = 4$). I.e., after the occurrence of pattern p (tuple 1), SWT can search until 4 tuples (tuples $2, 3, 4, 5$) looking for the itemset $\iota = \{G\}$.

STW starts looking tuple 2 and does not find ι, it is iteration (i) present in Figure 1(1). The search window expands covering the tuple 3, iteration (ii), and does not find the itemset $\{G\}$. The same happens to iteration (iii). At iteration (iv), the window has its maximum size allowed ($window = 4$), although it finds the itemset $\{G\}$. So the number of occurrences of pattern p' is one.

Now, consider the Situation 2 presented in Figure 1(2). In this situation, pattern $p =< (A\ B)\ D >$ is found

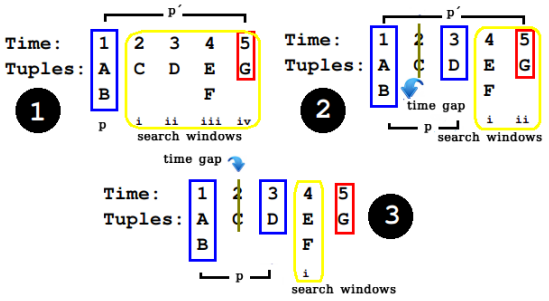


Figure 1: Three examples of the STW running.

just if $\mu \geq 1$ because pattern p presents one time gap between the itemsets $\{A\ B\}$ and $\{D\}$. Now, considering $\mu = 5$, pattern $p' =< (A\ B)\ D\ G >$ and $\iota = \{G\}$. To count how many times p' happens it is necessary to calculate the $numberOfTimeGaps$, as p presents one time gaps (tuple 2) $numberOfTimeGaps = 1$. So $\mu - numberOfTimeGaps = 4$. $\Delta(nextOccurence(p)) = 2$ because there is no other occurrence of pattern p, notwithstanding the database ends at tuple 5 and the distance between the last element of pattern p (itemset $\{D\}$) and the end of the database is 2 tuples. By that $window = 2$. STW checks tuple 3 (first one after pattern p) looking for item G and does not find it. The search window expands looking tuple 5 and finds item G. So the number of occurrence of pattern p' is one.

The Situation 3, presented in Figure 1(3), is almost the same one presented Situation 2. Nevertheless, consider now $\mu = 2$. Pattern p presents one time gap, so $numberOfTimeGaps = 1$. $\mu - numberOfTimeGaps = 1$ and $\Delta(nextOccurence(p)) = 2$ (as it was explained in Situation 2), so $window = 1$. STW search at tuple 4 and does not find G. As it is the maximum number of tuples, STW cannot check other tuples and STW ends. So the number of occurrence of pattern p' is zero (i.e. pattern p' does not happen to $\mu = 2$ in this database).

Experiments, Results and Discussion

We performed several experiments to validate our proposed approach. The experiment used a real database that comprises data from environment sensors placed in the area of Feijão River. Feijão is an important not polluted river that supplies water to the city of São Carlos, which is an important city in São Paulo state, Brazil. That database is used to compare MSTS and GSP by the performance and the semantic quality of the sequences found. To perform the experiments, a notebook with 8 GB of RAM memory, 500 GB of HD, processor Intel Due Core 2.53 GHz was used.

The data was collected since 1977 until 2004 and it keeps information about hydro–characteristics (the value of rainfall and the discharge of Feijão). The data presents granularity of days, i.e. the measures had happened daily. There are a large number of tuples in the database. A real example of tuple from *Feijão* River database is presented by Table 1. The data present the discharge rate in cubic meters per second and the rainfall rate in millimeters per hour. In order to simplify the notation, the units of measure are omitted in this work.

The database goes through a pre–processing step that discrete the data. To discretize the data set it was used Omega algorithm presented in (Ribeiro, Traina, and Traina 2008). Omega is a supervised algorithm that discrete numerical values in linear computational cost. It creates data intervals avoiding inconsistencies. Table 2 presents the same data in Table 1 after Omega processes.

After the pre-processing occurs, the data are grouped by weeks, aiming to reduce the granularity of the data. These grouping are made by calculating the average value for the rainfall and discharge of *Feijão* river. That also reduces the number of records to be mined.

On that kind of database, an event can influence another one if it occurs a time interval after it. So, at MSTS, the maximum value of this time gap is given by a new parameter, μ (mi). With the configuration of that parameter, we can solve two problems of this domain: gaps between occurrences of the events that compose patterns and the noises originated by measurement devices.

Our implementations of MSTS and GSP do not show the sub-patterns, that means: if there were two sequences, s_1 and s_2, with $s_1 \succeq s_2$ (s_2 is bigger than and composed by s_1), just s_2 will be showed to the user. The GSP algorithm is used as a baseline to compare with our method because there is no algorithm that implements similar windowing technique or something that has closer results to MSTS. Hence, MSTS is based on GSP. This experiment aims at exposing the gains and losses obtained with the implementation of the proposal method STW over GSP, producing the MSTS algorithm.

The fact that the algorithms do not present subpatterns makes sometimes the MSTS algorithm returns fewer sequences than GSP but the sequences found are bigger than those found by GSP. The database used has a critical point; at support greater or equal 46% both algorithms cannot find sequences bigger than 1 itemset and more than 3 sequence. Other interesting point is when support is 22%. At this point MSTS finds the same sequences to any sizes of *Stretchy Time Windows*. At this point, MSTS finds bigger and stronger itemsets, which GSP does not find. For that GSP presents bigger sequences with smaller itemsets. After this point, the number of sequence of GSP goes down until support value reaches 46%.

At Figure 2 there are three graphics comparing the results of the experiments. Graphic 2(a) presents a comparison by the number of sequences found. It is possible to see that GSP presents a good result, actually, after support value 22% the result is almost the same. That happens due to the configuration of the database. The sequences are not too strong, therefore, it is easier to find sequences with low support. With low support sometimes MSTS also finds fewer sequences because this implementation does not show the sub-patterns that we can see at Graphic 2(b), where the sequences are bigger. However, it is important to recall that bigger sequences encompass many smaller ones.

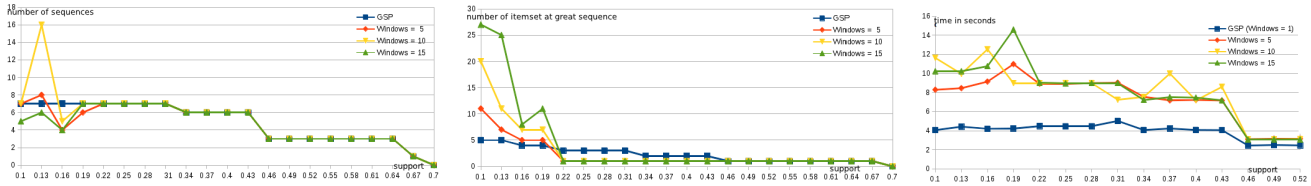
Graphic 2(b) is connected to Graphic 2(a). Whenever Graphic 2(b) has a fall, Graphic 2(a) has a rise. For the fact

Table 1: Example of tuples in *Feijão* River database.

Date	Discharge	Rainfall Rate
1979.03.19	$2.41m^3/s$	$1.2mm/h$
1979.03.10	$2.25m^3/s$	$0mm/h$
1979.03.21	$2.1m^3/s$	$0.6mm/h$

Table 2: Same tuples of Table 1 after pre-processing.

Tuple	Rainfall	Discharge
703	$Rainfall_4$	$Discharge_1$
704	$Rainfall_0$	$Discharge_1$
705	$Rainfall_2$	$Discharge_0$



(a) Number of sequences found (number of sequences) by support value.

(b) Number of itemsets on great sequences found (number of itemsets) by support value.

(c) Time used to run the implementation (seconds) by support value.

Figure 2: Comparison between MSTS and GSP over *Feijão* River database regarding the number of sequences found, the size of the biggest sequence found and the performance of the mining algorithm.

that a big pattern does not let sub-patterns appear and because this pattern does not exist, other stronger sub-patterns (greater support value) are presented. Graphic 2(a) also shows that with bigger μ value it is possible to find bigger sequences. And bigger sequences are important for that domain because it represents longer behaviors and, with that, it is possible to predict future behavior with higher time advance. I.e., MSTS provides greater analysis period of data behaviors.

At Graphic 2(c) the performance comparison is presented. MSTS has lost to GSP in all the tests, but it was expected; while MSTS is checking the support, it scans more tuples than GSP does. For that, MSTS presents a worse performance. On the other hand, changing the μ value, the performance does not change a lot. That means it is possible to adjust this value without influencing the performance.

The examples of patterns mined by GSP Algorithm presented are at Table 3. This patterns do not present time gaps. The first pattern is a 2-sequence (because it presents two itemsets) s_1^{gsp} whose frequency is 5.12%. I.e., in 5.12% of the database, the event $(Rainfall_0\ Discharge_0)$ (no rainfall and the measure of *Feijão* River discharge is a value less than 1.6157) happens immediately after (the next week after) the event $Rainfall_0$ (no rain). The second pattern mined by GSP Algorithm is a 3-sequence (sequence composed by three itemsets) s_2^{gsp}. That is frequent in 7.69%. That 3-sequence shows that there are periods of three weeks during which there is no rain.

Table 4 presents two examples of patterns mined by MSTS (where maximum value of *Stretchy Time Windows* is five weeks – $MSTS_5$) that have almost the same support values of that presented by Table 3 (patterns mined by GSP). The first pattern mined is a 2-sequence: $s_1^{MSTS_5}$, whose frequency is 5.12%. That first pattern shows that the event $Rainfall_5$ (rainfall rate between 3.2143 and 4.6286) happens at most five weeks after the event of $Discharge_2$ (*Feijão* River discharge rates between 2.13 and 2.62). So, this pattern can be presented as$< Discharge_2\ \Delta t_1\ Rainfall_5 >$ where $0 \leq \Delta t_1 \leq \mu$ and $\mu = 5$ weeks. The second pattern mined by $MSTS_5$ is a 3-sequence $s_2^{MSTS_5}$, whose frequency is 5.12%. That sequence shows that, after a while before the event $Discharge_6$ (*Feijão* River discharge rates between 3.3171 and 3.6286) has happened, $Rainfall_3$ (rainfall rate between 0.4857 and 1.6) happens. And after another period of time $Rainfall_0$ happens. So the sequence can be seem as $< Discharge_6\ \Delta t_1\ Rainfall_3\ \Delta t_2\ Rainfall_0 >$ where $0 \leq \Delta t_1 + \Delta t_2 \leq \mu$.

Table 5 contains two patterns mined by MSTS; to find these patterns the maximum of time gaps (μ) is 10 weeks, $MSTS_{10}$. That means that it can pass 10 weeks between the occurrence of an itemset and the next one in the sequences. The first pattern mined by $MSTS_{10}$ presented at Table 5 is a 2-sequence $s_1^{MSTS_{10}}$ whose frequency is 5.12% of the tuples in the database. That sequence is interpreted as: in at most ten weeks after the event $Rainfall_7$ (rainfall rate between 9.2571 and 15.3429), the discharge of *Feijão* River will be in the range $Discharge_7$ (*Feijão* River discharge rate between 3.6286 and 4.38). So, $< Rainfall_7\ \Delta t_1\ Discharge_7 >$ where $0 \leq \Delta t_1 \leq \mu$, and $\mu = 10$ weeks. The second pattern at Table 5 is a 3-sequence $s_2^{MSTS_{10}}$. Its frequency is 5.12% of the sequences in the database. That pattern presents a gradual decrease of the discharge value: first it is measured the *Feijão* River discharge in the range $Discharge_8$ (interval of $(4.38; 5.0771]$) and, then, after a Δt_1 weeks it is measure $Discharge_6$. Another Δt_2 weeks later, there is the event $Rainfall_5$. So, $< Discharge_8\ \Delta t_1\ Discharge_6\ \Delta t_2\ Rainfall_5 >$ where $0 \leq \Delta t_1 + \Delta t_2 \leq 10$ weeks. The pattern can take from three (because it is a 3-sequence) to thirteen weeks (because there is the time for the 3-sequence to happen plus the possible time intervals between the events, μ).

Table 6 presents two examples of patterns mined by MSTS where *Stretchy Time Windows Maximum Value* is

Table 3: Examples of patterns found by GSP over the *Feijão* River database.

Label	Pattern	Sup
s_1^{gsp}	$< Rainfall_0\ (Rainfall_0\ Discharge_0) >$	5.12%
s_2^{gsp}	$< Rainfall_0\ Rainfall_0\ Rainfall_0 >$	7.69%

Table 4: Examples of patterns found by MSTS over the *Feijão* River database. In this case, $\mu = 5$ (maximum value of *Stretchy Time Windows*): the patterns can present five weeks of time interval between their events.

Label	Pattern	Sup
$s_1^{MSTS_5}$	$< Discharge_2\ Rainfall_5 >$	5.12%
$s_1^{MSTS_5}$	$< Discharge_6\ Rainfall_3\ Rainfall_0 >$	5.12%

Table 5: Examples of patterns found by MSTS with $\mu = 10$: the patterns can present ten weeks of time interval between their events.

Label	Pattern	Sup
$s_1^{MSTS_{10}}$	$< Rainfall_7\ Discharge_7 >$	5.12%
$s_2^{MSTS_{10}}$	$< Discharge_8\ Discharge_6\ Rainfall_5 >$	5.12%

Table 6: Examples of patterns found by MSTS with $\mu = 15$: the patterns can present fifteen days of time interval.

Label	Pattern	Sup
$s_1^{MSTS_{15}}$	$< Discharge_8\ Rainfall_3 >$	5.12%
$s_2^{MSTS_{15}}$	$< Discharge_8\ Rainfall_5\ Rainfall_0 >$	5.12%

fifteen weeks, $MSTS_{15}$. That means that the patterns can present fifteen weeks of time gaps. The first example at the table is a 2-sequence frequent in 5.12% of the database. $< Discharge_8\ \Delta t_1\ Rainfall_3 >$ where $0 \leq \Delta t_1 \leq \mu$ and $\mu = 15$ (weeks). The interpretation of the pattern is: at most fifteen weeks after the value of *Feijão* River discharge has been $Discharge_8$, the river rate may be $Rainfall_3$. The second pattern shows a gradual decrease of the rainfall rate. The second sequence is the 3-sequence $< Discharge_8\ \Delta t_1\ Rainfall_5\ \Delta t_2\ Rainfall_0 >$ where $0 \leq \Delta t_1 + \Delta t_2 \leq 15$. So after Δt_1 weeks when it was measured $Discharge_8$, $Rainfall_5$ is measured. After another while (Δt_2 weeks), there is no rainfall ($Rainfall_0$). That shows a decrease of the rainfall after a high value of discharge of *Feijão* River. The pattern can take from three to eighteen weeks.

Conclusion

This work presented a new method to search sequential patterns over sparse and noisy data, which are the kind of data generated by environment sensors. The proposed method is called *Stretchy Time Windows* (STW), which allows all occurrences of a pattern to happen with gaps (spaces between the events of the sequence). The number of gaps is set by the user. The algorithm that implements the proposed method STW is called *Miner for Stretchy Time Sequences* (MSTS). The approach allows time gaps between events in sequences occurrences that aim to solve the problem of mining sparse patterns. Our experiments show that the proposed approach finds sequences with greater analysis time when compared with the previous methods. It is possible to see that the size of the *Stretchy Time Windows* influences the size of the sequences. With relatively low support values, it was possible to find sequences four and five times bigger than GSP. Furthermore, there are occasions, with low support, where we could find more sequences than GSP. As future work, MSTS will be modified for the application of incremental mining. This modification aims to improve the performance of the algorithm to check databases that present increments periodically. The incremental mining aims to keep the found patterns compatible with changes in old data or increments in new data.

Acknowledgments

We thank the Brazilian funding agencies CAPES, CNPq, FAPESP and FINEP for financial support and Federal University of Itabubá for the access on *Feijão* River database.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, 3–14.

Barseghian, D. *et al*. 2010. Workflows and extensions to the kepler scientific workflow system to support environmental sensor data access and analysis. *Ecological Informatics* 5:42 – 50.

Chen, H. 2010. Efficiently mining the recent frequent patterns over online data streams. In *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, 1 –4.

Han, J.; Pei, J.; and Yan, X. 2005. Sequential pattern mining by pattern-growth: Principles and extensions. *Foundations and Advances in Data Mining* 1:183–220.

Huang, T.-K. 2009. Developing an efficient knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases. In *New Trends in Information and Service Science, 2009. NISS '09. International Conference on*, 362 –371.

Masseglia, F.; Poncelet, P.; and Teisseire, M. 2009. Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Systems with Applications* 36:2677–2690.

Nunthanid, P.; Niennattrakul, V.; and Ratanamahatana, C. 2011. Discovery of variable length time series motif. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2011 8th International Conference on*, 472 –475.

Pei, J. *et al.*. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, 215–224. IEEE Computer Society.

Ribeiro, M. X.; Traina, A. J. M.; and Traina, Jr., C. 2008. A new algorithm for data discretization and feature selection. In *Proceedings of the 2008 ACM symposium on Applied computing*, 953–954.

Srikant, R., and Agrawal, R. 1996. Mining sequential patterns: Generalizations and performace improvements. In *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*, 3–17. Springer-Verlag.

Zabihi, F. *et al.*. 2010. Fuzzy sequential pattern mining with sliding window constraint. In *2nd International Conference on Education Technology and Computer*, V5–396 –V5–400.