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

Incremental Mining of Frequent Sequences in Environmental Sensor Data

Carlos R. Silveira Jr., Danilo C. Carvalho, Marilde T. P. Santos, Marcela X. Ribeiro
Department of Computing, Federal University of São Carlos
Washington Luís, km 235 – SP–310, 13565-905
São Carlos, São Paulo, Brazil

Abstract

The mining of sequential patterns in environment sensor data is a challenging task. Most of sequential mining techniques requires periodically complete data. Furthermore, this kind of data can be incomplete, present noises and be sparse in time. Consequently, there is a lack of methods that can mine sequential patterns in sensor data. In this paper, we proposed IncMSTS, an incremental algorithm for mining stretchy time patterns. The proposed algorithm is an incremental version of the MSTS, a previous not scalable algorithm that mines stretchy time patterns in static databases. The experiments show that IncMSTS runs up to 1.47 times faster than MSTS. When compared to GSP, the literature baseline algorithm for mining frequent sequences, IncMSTS can return 2.3 more sequences and the returned sequences can be 5 times larger, indicating that the sparse time analysis promoted by IncMSTS broadens the mining potential of finding patterns.

Introduction

The task of sequential mining in environment sensor data is a difficult issue because of their spatio-temporal, incompleteness, noisy and sparse characteristics. In general, we have the presence of time sparse patterns, presenting time gaps inside. These patterns are not detected by traditional sequential mining techniques.

The first algorithm that mines sparse patterns in sensor data was MSTS (Silveira Junior, Prado Santos, and Ribeiro 2013). The MSTS algorithm works varying the time analysis of the sequences occurrences and performing a greedy scan in the data space to search for patterns. Consequently, MSTS has a high computational cost. In fact, the mining of patterns in flexible time scale requires increasing the searching space of the mining algorithm, making it almost unfeasible to work with real applications, e.g. sensor data monitoring, which requires online analysis of periodically incremented data. In this paper, we increase the applicability of the MSTS algorithm, speeding up its performance in a incremental manner, allowing it to analyze real sensor data.

Background and Related Work

A sequential pattern is a sequence of events (itemsets) [1] that keeps an order among their occurrences. Let's consider a sequence $s =< i_1 \ldots i_n >$ 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 $i_k \in s$, an $i'_k \in s'$ where $i'_k \subset i_k$. The traditional sequential mining algorithms usually separate the sequences in frequent and non-frequent ones. The frequency of a sequence is called support, it can be calculated as $support(s) = \frac{|number\ of\ occurrence\ of\ s|}{|number\ of\ sequences\ on\ the\ database|}$ and it has a value between zero and one ($[0; 1]$) (Huang 2009). If $support(s)$ is equal or greater than a minimum support value ($minSup$) set by the user then the sequence s is considered frequent.

Incremental Data Mining is a technique that aims to avoid reprocessing the whole database every time the data is updated (Chiu et al. 2011). It shows good performance results working in data streams (that suffers constantly updates). On the other hand, the precision of the algorithm can be degraded because of the possibility of error propagation. The first algorithm, IncSpan proposed by (Cheng, Yan, and Han 2004), presented the Semi–Frequent Sequences strategy.

The Semi–Frequent Sequences strategy stores also semi–frequent patterns [2] because they are considered promising to become frequent after the update of the database. However, IncSpan does not find sparse patterns and it also receives as input a horizontal database. The proposed algorithm, IncMSTS, applies the Semi–Frequent Sequences strategy as IncSpan and works with time series dataset to find sparse patterns.

(Niranjan et al. 2011) presents Modified IncSpan. This approach uses a new proposed structure called Patricia to generate and organize the patterns. Such structure can handle increments INSERT (sequences insertion) and APPEND (new items insertion). The Modified IncSpan presents the same restriction of IncSpan.

In (Silveira Junior, Prado Santos, and Ribeiro 2013), the

[1]The word "events" is sometimes used instead of "itemsets" because "event" is more user friendly to explanations focusing in the application domain.

[2]A semi-frequent pattern has its support value in $[minSup \times \delta; minSup]$ where δ-value (in $[0; 1]$) is set by a domain expert.

concept of Stretchy Time Patterns (STP) extraction is introduced. In this paper, the window-based mining strategy of IncMSTS was presented.

The Proposed Algorithm: *Incremental Miner of Stretchy Time Sequences*

The *Incremental Miner of Stretchy Time Sequences* (IncMSTS) algorithm is a new incremental algorithm that finds Stretchy Time Patterns (STP). IncMSTS is based on MSTS (Silveira Junior, Prado Santos, and Ribeiro 2013). STP is a pattern that presents time gaps between its events (sparse pattern) and it is defined by Equation (1), where $i_1 \dots i_n$ are itemsets (not necessarily disjoint) and $\Delta t_1 \dots \Delta t_{n-1}$ are time gaps.

$$s = < i_1 \; \Delta t_1 \; i_2 \; \dots \; \Delta t_{n-1} \; i_n > \quad (1)$$

For each occurrence *occ* of a sequence s, the total of time gaps cannot be greater than μ (Maximum Time Gap Size – parameter set by user), i.e. $[\sum_{k=1}^{n-1} \Delta t_k^{occ}] \leq \mu$.

The IncMSTS algorithm, presented in Algorithm 1, receives as input: (i) Set of data increment *db*, which is the data increment that the dataset has received, if it is the first time, *db* will be the whole dataset. (ii) Minimum support value $minSup$ that defines the frequent patterns. (iii) Maximum Time Gap Size μ that defines how sparse a pattern can be. (iv) δ value that defines the semi-frequent patterns. (v and vi) Old frequent and semi–frequent patterns, fs and sfs.

IncMSTS finds the frequent itemsets in the data increment (line 1) and then these itemsets rebuild the old frequent and semi–frequent patterns (line 2). The rebuilding of the old pattern consists in marking their occurrences in the data increment *db*, if it is the first time the dataset is processed this step is not executed. The data increment *db* are mined aiming to find new frequent patterns (line 3 up to 5); if it is the first time the dataset is processed this step finds the frequent and semi-frequent patterns. Then the old knowledge (fs and sfs), which is rebuilt to the new dataset, is mined (line 6 up to 15) – if there is no old knowledge it is not executed. To process the old knowledge, the support of each pattern is recalculated (line 7) considering the old occurrences and the new ones. Then, if the new support is equal to or greater than $minSup \times \delta$, i.e. the pattern is at least semi–frequent (line 8 up to 14), it is rechecked (line 9): if the pattern remains frequent, it is generalized (line 10), otherwise it is added in the new semi-frequent set (line 12).

Algorithm 1: The *Incremental Miner of Stretchy Time Sequences* Algorithm.

```
   Input: Set of data increment $db$, $minSup$, $\mu$, $\delta$, set of frequent sequences $fs$, set of
          semi–frequent sequences $sfs$.
   Output: Set of patterns $C$, set of semi–frequent patterns.
1  $C \leftarrow \{frequent\, itemset\} \in db$ ;
   /* Recreating the old pattern using the data increment.          */
2  renewPatterns($fs \bigcup sfs$, $db$);
   /* Finding new patterns in the data increment.                    */
3  foreach pattern $p \in C$ do
4      $C \leftarrow C \bigcup generalize(p, db, minSup, \mu, \delta)$ ;
5  end
   /* Updating old patterns.                                         */
6  foreach pattern $p \in fs \bigcup sfs$ do
       /* Calculating support considering the data increment.        */
7      $sup_p \leftarrow \frac{numberofOccurence(p)}{Total\, of\, sequences}$ ;
       /* Classifying patterns in frequent or semi-frequent.         */
8      if $sup_p \geq minSup \times \delta$ then
9          if $sup_p \geq minSup$ then
10             $C \leftarrow C \bigcup generalize(p, bd, minSup, \mu, \delta)$ ;
11         else
12             addingSemiFrequent(p) ;
13         end
14     end
15 end
```

Algorithm 2: The *Generalize* Function algorithm.

```
   Input: Pattern $p$, dataset $db$, $minSup$, $\mu$, $\delta$.
   Output: Set of patterns derived from $p$ $Results$, semi–frequent patterns.
1  $Itemsets \leftarrow \{frequent\, itemsets\} \in db$ ;
2  $Result \leftarrow \{p\}$ ;
   /* Combining pattern and frequent itemsets, and cheking
      support.                                                       */
3  foreach pattern $p' \in Result$ do
4      foreach itemset $\iota \in Itemsets$ do
           /* Calculating support of $< p' \; \iota >$.              */
5          $sup_p \leftarrow \frac{checkingOccurrence(p', \iota, \mu)}{Total\, of\, sequences}$ ;
           /* Sorting $< p' \; \iota >$ in frequent and semi-frequent.  */
6          if $sup_p \geq minSup \times \delta$ then
7              if $sup_p \geq minSup$ then
8                  $Results \leftarrow Results \bigcup \{p' \bigcup \iota\}$ ;
9              else
10                 $addingSemiFrequent(p' \bigcup \iota)$ ;
11             end
12         end
13     end
14 end
```

The *generalize* function is called twice in Algorithm 1, at lines 4 and 10. The function *generalize* is presented by Algorithm 2. Its inputs are: a pattern p that will be generalized, dataset *db*, minimum support value $minSup$, size of maximum time gap μ and δ value. The outputs are: set of generated frequent patterns and the semi-frequent patterns.

generalize finds the frequent itemsets in the dataset *db* (line 1). Then the algorithm combines pattern p with each frequent itemset (line 3 up to 14). The support of each combination is calculated (line 5) and the combination is sorted into semi-frequent (line 10), frequent (line 8) or non-frequent (the latter combination is discarded). The frequent combinations go to the $Result$ set and, in another iteration of the loop at line 3, the combinations are recombined (with frequent itemsets) trying to create bigger sequences. In this way, all patterns whose prefix is pattern p are generated. The *generalize* procedure ends when no pattern in $Result$ can generate a bigger pattern.

The function *checkingOccurrence* is called by Algorithm 2 (line 5). Function *checkingOccurrence* implements *Strechy Time Window* (STW) method from MSTS algorithm. The noisy data can be solved by STW because the noise can be considered as moments where nothing happens; time gaps.

STW method uses the parameter μ, which defines the maximum size of a time gap. The STW algorithm and an example are presented in (Silveira Junior, Prado Santos, and Ribeiro 2013).

An optimal window size depends on the domain that IncMSTS is being applied. The setting of μ value is made by a domain expert. IncMSTS assigns flexibility to the algorithm and can be used in any domain that takes advantage of its features, i.e. inconsistent and incremental behavior data. As shown by the experiments, the window size does not have a critical role in the performance of the algorithm.

The IncMSTS algorithm has a quadratic complexity in

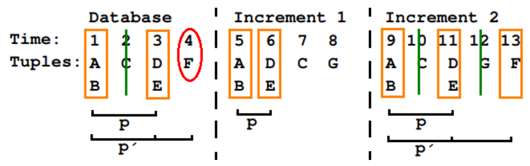

Figure 1: Example of IncMSTS running in an incremental dataset that has received three increments.

function of the size of the dataset ($\Theta(n^2)$, where n is the number of tuples in the dataset), as well as GSP, MSTS and any GSP base algorithm. The performance improvement happens because the incremental technique eliminates the need of reprocessing the whole dataset when it is an evolutionary dataset, i.e. fewer tuples are processed.

Examples of IncMSTS Processing

Consider the situation presented in Figure 1: it is the original data set ($Dataset$) composed by 4 tuples. The data have suffered two data increments ($Increment$ 1 and $Increment$ 2). $Increment$ 1 inserts 4 tuples and $Increment$ 2 inserts 5 tuples in $Dataset$. To perform the explanation of this example, consider $\mu = 2$ (the value of the maximum search window is 2 time units) and $\delta = 0.5$ (to be considered a semi-frequent pattern, its support value should be greater than or equal half of minimum support value).

Now consider the situation without any increment. Consider stretchy time pattern $p =< (A\ B)\ (D\ E) >$ that is frequent with any $\mu \geq 1$ because it has happened once (at tuples timestamps $\{1, 3\}$) and there is a time gap in this occurrence (at tuple timestamps 2). Stretchy time pattern p' is composed by pattern $p \bigoplus \{F\}$ (pattern p concatenated item F). So the goal is to find in $Dataset$ occurrences of $\{F\}$ that happen after the occurrences of pattern p and are covered by *Search Time Window* (method STW). The method STW receives the pattern p and the item F. As it was explained in STW algorithm (Silveira Junior, Prado Santos, and Ribeiro 2013), to the only occurrence of pattern p in $Dataset$, the size of *Search Time Window* is one time unit (because there is a time gap in the occurrence of pattern p and the *Maximum Search Time Window* is two time units). That way, the algorithm searches for the occurrence of itemset $\{F\}$ in the next "size of *Search Time Window*" tuples after the pattern p occurrence (in this case, just the next tuple whose timestamps is 4). So, at tuple 4, the algorithm finds F and, then, the pattern p' is considered frequent. Finally, IncMSTS returns the patterns p and p' both frequent.

After the $Increment$ 1 of data the support of the frequent and semi-frequent pattern are rechecked. Considering that, with increment of data, a frequent pattern should happens twice and a semi-frequent pattern should happen just once. Algorithm 1 finds the old patterns p and p' in the new data ($Increment$ 1). There is an occurrence of pattern p in the tuples $\{5, 6\}$, although there is no occurrence of pattern p'. This process uses STW method because the new occurrences can be sparse too. As pattern p happens twice (one time in the $Dataset$ and another time in the $Increment$ 1), it is considered frequent and still at FS (set of frequent sequences). As pattern p' has happened just one time (just in the $Dataset$); it is considered semi-frequent; it is removed from FS and added in SFS (set of semi-frequent sequences). Thereby IncMSTS returns just the pattern p in this iteration.

With the last increment of data ($Increment$ 2), the support of the frequent and the semi-frequent sequences are rechecked. Considering that a frequent pattern should happen twice and a semi-frequent pattern should happens just once, as the situation of $Increment$ 1. IncMSTS recreates the old sequences ($FS \bigcup SFS$) using the data increment (in $Increment$ 2). As function *renewPatterns* (Algorithm 1, line 2) also implements STW method, it can find sparse occurrences of the patterns in the data increment. So pattern p happens in tuples $\{9, 11\}$ with a time gap (tuple 10) and pattern p' happens in tuples $\{9, 11, 13\}$ with two time gaps (the maximum allowed by μ parameter). That way, p happens three times (twice in $Dataset \bigcup Increment$ 1 and once in $Increment$ 2) and it is considered frequent. Furthermore, the semi-frequent sequence p' happens twice (once in $Dataset \bigcup Increment$ 1 and another time in $Increment$ 2) and it is promoted to a frequent sequence. Therefore IncMSTS returns the patterns p and p' as frequent.

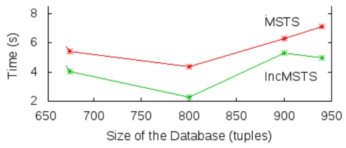

Figure 2: Performance comparison between IncMSTS and MSTS with the evolution of the dataset.

Table 1: Example of patterns found in the database without increment.

Label	Pattern	Support
s_1	$< (Rainfall_0\ Discharge_1)\ (Rainfall_0\ Discharge_1) Rain\_ fall_0 Discharge_1\ Discharge_1\ Discharge_4\ Discharge_0 >$	0.06
s_2	$< (Rainfall_0\ Discharge_0)\ Rainfall_0 Rainfall_0\ (Rainfall_0\ Discharge_0) Discharge_4\ Discharge_0 >$	0.06

Experiment and Results

We performed several experiments to validate our proposed approach, but here we present the ones performed with the same dataset employed in (Silveira Junior, Prado Santos, and Ribeiro 2013) for keeping consistence between both works. Furthermore, either pre–processing step or input configuration ($\mu = 15$ weeks, $minSup = 5\%$ and $\delta = 0.5$) still the same.

The performance comparison is presented in Figure 2, IncMSTS has performed faster (until 1.47 times) than MSTS. Empirical comparison shows that IncMSTS returns the same MSTS sequence. Thus, IncMSTS has 100%-precision value as MSTS because both algorithms make full scan search over the data. As example, Table 1 presents an example of two STP's; Table 2 and Table 3 present the evolution of these patterns because of data increase.

The first pattern in Table 1 is s_1, whose support value is 0.06. This pattern starts the double happening of the itemset ($Rainfall_0\ Discharge_1$) that are followed by the measure

Table 2: Example of patterns found in the database after the first increment.

Label	Pattern	Support
$s_{1.1}$	$< (Rainfall_0\ Discharge_1)\ Rainfall_0\ Discharge_1 >$	0.05
$s_{1.2}$	$< (Rainfall_0\ Discharge_1)\ Discharge_1 >$	0.07
$s_{2.1}$	$< (Rainfall_0\ Discharge_0)\ Rainfall_0\ Discharge_0 >$	0.07
sn_1	$< Discharge_7\ Rainfall_3\ (Rainfall_3\ Discharge_6) >$	0.03

Table 3: Example of patterns found in the database after the second increment.

Label	Pattern	Support
$s_{1.1.1}$	$< (Rainfall_0\ Discharge_1)\ Rainfall_0 >$	0.0512
$s_{1.2}$	$< (Rainfall_0\ Discharge_1)\ Discharge_1 >$	0.0598
$s_{2.1}$	$< (Rainfall_0\ Discharge_0)\ Rainfall_0\ Discharge_0 >$	0.059
sn_2	$< (Rain_5\ Discharge_3\ Autumn)\ (Rain_5\ Discharge_8) >$	0.008

$Rainfall_0$ (no rain) and twice the measure of $Discharge_1$ (discharge of Feijão River value in $[2.13; 2.62)$) all separated. After that, it happens $Discharge_4$ (discharge value in $[3.0514; 3.3171)$) and $Discharge_0$ (discharge value in $[1.6157; 2.13)$). Between all itemset could exist Δt's. And for each occurrence of s_1, the $[\sum_{k=1}^{6} \Delta t_k] \leq \mu$, it means that the value of any Δt_p ($0 \leq p \leq k$) can be different for each occurrence of the pattern.

Pattern s_2, in Table 1, is a 6-sequence (it has six itemsets) whose support is 0.06. $s_2 =< (Rainfall_0\ Discharge_0)\ \Delta t_1\ Rainfall_0\ \Delta t_2\ Rainfall_0\ \Delta t_3\ (Rainfall_0\ Discharge_0)\ \Delta t_4\ Discharge_4 \Delta t_5\ Discharge_0 >$.

Both patterns presented in Table 1, $s_{1,2}$ evolved to smaller patterns with the first increment of data. Table 2 brings the evolved patterns; pattern s_1 evolves to patterns $s_{1.1}$ and $s_{1.2}$ and pattern s_2 evolves to pattern $s_{2.1}$.

The pattern $s_{1.1}$, Table 2, is originated from pattern s_1. Pattern s_1 is not frequent with the increment. However, its sub-pattern $s_{1.1}$ is frequent, albeit with lower support value. The pattern $s_{1.2}$ also originated from pattern s_1 and it is a 2-sequence whose support is higher than s_1 support ($support(s_{1.2}) = 0.07$).

Pattern $s_{2.1}$ (Table 2) evolved from pattern s_2 (Table 2). $s_{2.1}$ presents fewer itemsets (it is 3-sequence), with higher support values.

The pattern sn_1 is called Current Pattern (CP). This pattern reached the minimum support if only the tuples of last database increment is considered. Such patterns may show a new trend of the dataset that must be observed. In the first runtime, all patterns mined are current patterns.

$sn_1 =< Discharge_7\ \Delta t_1\ Rainfall_3\ \Delta t_2\ (Rainfall_3\ Discharge_6) >$ whose support is 0.03 less than $minSup$ but it would be superior if just the increment were mined.

Pattern $s_{1.1.1}$ (Table 3) evolved from pattern $s_{1.1}$ (Table 2). $s_{1.1.1}$ lost the $s_{1.1}$ last itemset $Discharge_1$ and increased its support value (0.0512). With the second increment, $s_{1.1}$ is no more frequent, although $s_{1.1.1}$ (sub-pattern) remains frequent. The patterns $s_{1.2}$ and $s_{2.1}$, presented in Table 3, have not suffered any evolution with the second data increment and they are still frequent (from Table 2). However, their support values have been decreased: $s_{1.2}$ decreased 14.57% and $s_{2.1}$ decreased 15.71%. The CP sn_1 (Table 2) ceases to be frequent after the increment in the dataset. A new CP happens with the second increment, sn_2, presented in Table 3. sn_2 is the sequence $< (Rainfall_5\ Discharge_3\ Autumn)\ \Delta t_1\ (Rainfall_5\ Discharge_8) >$ whose support value is 0.8%. It means that, considering only the increment, sn_2 is frequent (its support value is greater or equal than $minSup$), although considering all the sequences, sn_2 is not frequent (support less than $minSup$).

Conclusions

The sequential patterns extraction in environment sensor datasets is achallenge. The domain present its own characteristics, e.g. spatio–temporal data, periodic increments of data, noises and patterns that can present time gaps between events (sparse patterns). These datasets contain knowledge that can be extracted by a data mining algorithm. However, there is no state of art algorithm that considers all these particular characteristics. Hence, we proposed the *Incremental Miner of Stretchy Time Sequence* (IncMSTS) algorithm that implements the method *Stretchy Time Windows* (STW). STW is used to mine patterns in datasets with noises; the noise are considered as common time gaps. The IncMSTS algorithm implements incremental data mining technique that stores semi–frequent patterns. This technique improves performance in incremental datasets. Our experiments have shown that IncMSTS returns sequences up to 5 times larger and up to 2.3 times in higher number than GSP algorithm. Furthermore, IncMSTS could run 1.47 times faster than its non-incremental version (MSTS).

Acknowledgments

We thank CAPES, CNPq, FAPESP and FINEP for the financial support and Federal University of Itajubá for the database access.

References

Cheng, H.; Yan, X.; and Han, J. 2004. Incspan: incremental mining of sequential patterns in large database. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD 04, 527–532. New York, NY, USA: ACM.

Chiu, S.-C.; Li, H.-F.; Huang, J.-L.; and You, H.-H. 2011. Incremental mining of closed inter-transaction itemsets over data stream sliding windows. *Journal of Information Science* 37(2):208–220.

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.

Niranjan, U.; Krishna, V.; Srividya, K.; and Khanaa, V. 2011. Developing a dynamic web recommendation system based on incremental data mining. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 3, 247–252.

Silveira Junior, C. R.; Prado Santos, M. T.; and Ribeiro, M. X. 2013. Stretchy time pattern mining: A deeper analysis of environment sensor data. In *The Twenty-Sixth International FLAIRS Conference*, 468–468.