World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:12, No:9, 2018

A Comparative Study of GTC and PSP Algorithms for Mining Sequential Patterns Embedded in Database with Time Constraints

Safa Adi

***Abstract*—**This paper will consider the problem of sequential mining patterns embedded in a database by handling the time constraints as defined in the GSP algorithm (level wise algorithms). We will compare two previous approaches GTC and PSP, that resumes the general principles of GSP. Furthermore this paper will discuss PG-hybrid algorithm, that using PSP and GTC. The results show that PSP and GTC are more efficient than GSP. On the other hand, the GTC algorithm performs better than PSP. The PG-hybrid algorithm use PSP algorithm for the two first passes on the database, and GTC approach for the following scans. Experiments show that the hybrid approach is very efficient for short, frequent sequences.

***Keywords*—**Database, GTC algorithm, PSP algorithm, sequential patterns, time constraints.

I. Introduction

THE stored data has become very large; for this reason the researchers give more attention to convert this huge volume of stored data into helpful information for different using [5]. Based on this the problem of miming patterns has large interest these days [4]. Pattern recognition has many important applications like analyzing the DNA sequences and the customer behavior [14].

Many algorithms are proposed for mining. "Broadly data mining algorithms are classified into two categories as Pattern-Growth approach or candidate generation and a Prior-Based" [1]. By introducing constraints such as user-defined threshold, user-specified data, minimum gap or time, algorithms outperforms better [1], [11].

Sequential Pattern is a special case of structured data mining, that is concerned with finding statistically relevant patterns between data examples, where the values are delivered in a sequence. It is usually discrete values; thus time series miming is closely related. "Sequential Pattern Mining is an efficient technique for discovering recurring structures or patterns from very large dataset" [13], [18].

The concept of sequential patterns is introduced to capture typical behaviours over time. For example, from a customer purchase database, a sequential pattern can be used to develop marketing and product strategies. For effectiveness considerations, constraints become more and more essential in many applications. Nevertheless, handling time constraints are far away for trivial; since handling such these constraints have a lot of expensive operations; to examine if these constraints are verified on sequences [1], [2], [12].

Safa Adi is with the Department of Computer and IT, Palestine Polytechnic University, Palestine, Hebron (e-mail: $safa\_adi@ppu.edu$).

So we can address two main problems when the handling time constraints in mining sequential patterns in the database. The first problem: is it possible to enhance traditional level wise algorithms to handle time constraints? The second problem: is it possible to consider time constraints directly in the mining process rather than a post-processing step? [1], [10].

This paper will compare two algorithms which are for mining generalized sequential patterns with time constraints in large databases. The first algorithm is GTC (Graph for Time Constraints), the main new feature of this algorithm is it handled the time constraints during the mining process and separately from the counting step of sequence of data [1]. The second algorithm is PSP (Prefix-tree for Sequential Patterns), it used different data structure as intermediate step [2].

II. Sequential Patterns and Time Constraints

In this section, we first formulate the concept of sequences and then look at the time constraints. Let DB be a database of customer transactions where each transaction T consists of [15], [16]: The customer number, has the symbol by Cid, the transaction time has the symbol time and the set of items or itemset involved in the transaction, its symbol is it.
The required definitions are [1], [2]:-

- Sequence: Let $I = \{i_1, i_2, ..., i_m\}$ be a finite set of items. The itemset is a non-empty set of items. The sequence S is a set of itemsets ordered according to their time-stamp. It has the symbol as $(it_1, it_2, ..., it_n)$ where $it_j, j \in 1...n$, is an itemset. A k-sequence is a sequence of k-items. Performing this task requires providing any data sub-sequence S in DB with a support value giving its number of occurrences in DB [1].
- Support: A sequence transaction database is a set of sequences The support of a sequence S in a transaction database DB [1], denoted Support(S, DB), is defined as Support(S, DB) = $\left|\left\{C \in DB | S^1 C\right\}\right|$. The frequency of S in DB is Support $(S, DB) / |DB|$.
- Frequent Sequential Pattern Problem: Given a user-defined minimal support threshold, denoted σ, the problem of mining sequential patterns is to find sequences S in DB such that Support $(S, DB) \geq \sigma$. Such sequences are called frequent [1].
- Generalized Sequences: Given a user-specified minimum time gap (minGap), a maximum time gap (maxGap) and a time window size (windowSize), a data sequence

$d = (d_1, d_2, ..., d_m)$ is said to support a sequence $S = (s_1, s_2, ..., s_n)$ if there exist integers $l_1 \leq u_1 < l_2 \leq u_2 < ..... < l_n \leq u_n$, such that [1]:

- s_i is contained in $\bigcup u_i k = l_i, 1 \leq i \leq n$.
- transaction-time (du_i) - transaction-time $dl_i \leq ws, 1 \leq i \leq n$.
- transaction-time (dl_i) - transaction-time $(du_i - 1) > minGap, 2 \leq i \leq n$.
- transaction-time (du_i) - transaction-time $(dl_i - 1) > maxGap, 2 \leq i \leq n$.

- Frequent Generalized Sequential Pattern Problem: Given a user-defined minimal support threshold, denoted σ, user-specified minGap and maxGap constraints, and a user-specified sliding windowSize, the problem of generalized sequential pattern mining is the extraction of sequences S in DB such that Support $(S, DB) \geq \sigma$ [1].

III. Algorithms for Mining Sequential Patterns with Time Constraints

The step of finding all frequent sequences is a challenge because the search domain is huge: let $(s_1, s_2, ..., s_m)$ be a provided sequence and $n_i = |s_j|$ cardinality of an itemset. Then the search domain, i.e. the set of all possible frequent sequences is $2n_1 + n_2 + ... + n_m$ [1]-[4].

Many novel approaches for sequential pattern mining were proposed, such as Apriori, AprioriALL, GSP, SPADE, SPAM, and PrefixSpam [3]. In this section considers the (GSP, GTC, PSP) algorithms for mining sequential patterns with time constraints in large databases. Section III.D in will explain the hybrid approach that analyzes the performance of PSP with GTC.

A. GSP (Generalized Sequential Pattern) Algorithm

GSP was the first Apriori-based approach (or levelwise approach), which is an alternative family of data mining algorithms; that used breath first manner [17]. To build up candidates and many sequences, the GSP algorithm makes multiple passes over the database [6], [9].

The first step aims at computing the support of each item in the database. When this step has been completed, the various items (i.e. those that satisfy the minimum support) have been discovered. They are considered as normal 1-sequences (sequences having a single itemset, itself a singleton). The set of candidate 2-sequences is determined according to the following assumption: candidate 2-sequences could be any two different items, whether embedded in the same transaction or not. Frequent 2-sequences are built by counting the support. Based on this point, candidate k-sequences are generated from frequent (k-1)-sequences obtained in pass-(k-1).

The main idea of candidate generation is to retrieve, from among (k-1) sequences, pairs of sequences (S, S') such that discarding the first element of the former and the last element of the latter results in two fully matching sequences. When such a condition holds for a pair (S, S'), a new candidate sequence is built by appending the last item of S' to S.

Then they computed the support for all possible sequences, the sequence with minimum value become various. This process is repeated until all candidate sequences are finished. After this step, all candidate sequences are arranged by using a hash-tree data-structure [1], [2].

The leaves of the tree has all possible sequences, on the other hand the intermediate nodes will have the hash tables. Then each sequence is hashed to find the candidates in data. When browsing a data sequence, time constraints must be managed. This is performed by navigating downward or upward through the tree, resulting in a set of possible candidates. For each candidate, GSP checks whether it is contained in the data sequence. GSP causes bottlenecks because it: scans the database multiple times, and generates a huge set of candidate sequences; furthermore, it is inefficient for mining long sequential patterns. So there is a need for more efficient mining methods [1], [2].

B. PSP (Prefix-Tree for Sequential Patterns) Algorithm

This approach resumes the general principals of GSP but, it makes use of a different intermediary data structure for organizing candidate sequences, to improve the efficiency of retrievals.

In PSP algorithm, at each step k, the DB is browsed for counting the support of the current candidate. Then the common sequence set Lk can be built. From this placed new candidates are exhibited for being dealt at the next step. The algorithm stops when the most extended frequent sequences, embedded in the DB are discovered thus the candidate generation yields an empty set of new candidates. Support is a function giving for each candidate its counting value stored in the tree structure.

The tree structure is managed by an algorithm as follow: at the k_{th} step, the tree has a depth of k. It captures all the candidate k-sequences in the following way. Any branch, from the root to the leaf stands for a candidate sequence and considering a single branch, each node at depth l(k¿=l) captures l_{th} item of the sequence. Furthermore, along with an item, a terminal node provides the support of the sequence from the root to the considered leaf (included). Transaction cutting is captured by using labeled edges [2].

The algorithm is as follow:-

Input: T the tree containing all candidates and frequent sequences,
a data sequence d and its sequence identifier idseq.
The step k of the general algorithm.
Output: T the set of all candidate sequences contained in d.

```
l_a=FirstItemSet(d).item();
while (l_a<=LastItemSet(d).time())  do
        u_a=l_a;
while ((u_a-l_a)<ws) do
        l_p={i_p ∈ d/i_p.time() ∈[l_a,u_a]};
        for each i_p ∈I_p do
                if (i_p ∈root.Children) then
                        depth=0;
                        FindSequence(l_a,u_a,root.children(i_p),i_p,d,idseq,depth);
                u_a=(u_a.succ()).time;
        l_a=(l_a.succ()).time();
```

Fig. 1 PSP algorithm for candidate verification [2]

During the candidate verification phase in PSP prefix-tree, all terminal nodes (at depth k) which are brothers stand for continuations of a typical (k-1) sequence. Thus it is costly and not necessary to examine this common sequence for all k-sequences extending it. Retrieving candidates means poor navigation through the tree. Once a leaf is reached, the single operation to be performed is incremented the support value [1], [2].

The possible sequences will be examined, then the are deleted to prune the tree and minimize the used memory space. All leaves not satisfying a threshold support are removed. When such deletion complete, the tree no longer captures candidate sequences but instead frequent sequences [2], [7].

C. GTC (Graph for Time Constraints) Algorithm

This algorithm solved the following problem: how to reduce the time required for comparing a data sequence with the candidate sequences? This problem can be explained by recalculating a relevant set of sequences to be tested for a data sequence. By recalculating this set, the algorithm will be more efficient in analyzing the data sequences in the following two ways:

- The navigation through the candidate sequence tree does not depend on the time constraints defined by the user.
- This navigation is only performed on the most extended sequences, that is to say on sequences not included in other sequences.

This approach takes up all the fundamental principles of level wise algorithms. It contains many iterations. The iterations start at the size-one sequences and, at each iteration, all the frequent sequences of the same size are found. Moreover, at each iteration, the candidate sets are generated from the frequent sequences found at the previous iteration [1].

The main new feature of GTC algorithm which distinguishes it from traditional level wise algorithms is that handling of time constraint is done before and separate from the counting step of a data sequence. So, Upon reading a customer transaction d in the counting phase of pass k, GTC has to determine all the maximal combinations of d respecting time constraints. This means; that GTC algorithm has to identify all the k-candidates supported by the maximal sequences issued from GTC iterations on d and increment the support counters associated with these candidates without considering time constraints any more.

In [1] they defined the GTC algorithm as follow:-

```
Function GTC
Input: a data sequence d
Output: the sequence graph Gd(V,E)
Gd(V,E)=GTCws(d);
foreach item i ∈ Gd(V,E) do
        foreach item j∈ Gd(V,E) do
                if J.isPrec[i]=1 OR j ∈ i.next then
                        addMax(i,j);
                        //Adds j to the pointer list for the j valued cell associated to i
End function GTC
```

Fig. 2 GTC algorithm for minGap, windowSize and maxGap [1]

D. Hybrid Approach (PG-Hybrid)

We designed some experiments to illustrate that, for the sake of efficiency, the time required for building the sequence graph must be less than the difference between the time for candidate verification in PSP and candidate verification in GTC. First, to find the time for building the sequence graph, we carried out some experiments to analyze the performance of PSP with GTC when mining sequential patterns without time constraints, even if GTC is less efficient than PSP, the performance gap between the two algorithms is still constant [8], [20].

It is noticeable that even when considering low values for time constraints (for example if windowSize or minGap was set to 1), in the second step for the testing candidate [1], PSP was more efficient. Nevertheless, despite this efficiency, GTC was faster for the whole process. To take advantage of this behaviour, we defined a new algorithm, called PG-Hybrid, using the PSP approach for the two first passes on the database and GTC for the following passes [1], [2].

IV. Comparing SPS Algorithm with GTC Algorithm

In GSP, because of the defined tree structure, all candidates sequences are preserved and entirely stored in the leaves, so a massive set of candidates will be generated. But in PSP algorithm, the organization of candidates sequences according to their common elements, this means that the initial sub-sequences common to several candidates are stored only once. The prefix-tree structure that used in PSP is more efficient than the hash-tree structure used in GSP algorithm [19]. So PSP is less costly from a memory viewpoint. GTC algorithm is faster than GSP; because the time constraint is carried out before and separately from the counting step of the data sequence.

PSP algorithm, use the prefix-tree structure for sequences. This algorithm takes advantages of its semantics for avoiding useless and costly operations when verifying candidates. On the other hand GTC algorithm, considered the problem of discovering sequential patterns by handling time constraints [15], [18]. Instead of mining these constraints on the fly during the data sequence analysis, it attempted to use time constraint preprocessing for efficiently identifying various generalized sequences. In which time constraints are handled in the earlier stage of the algorithm to provide significant benefits. GTC use to build a sequence graph biased on time constraints.

As illustrated in Table I, we can see that GTC algorithm is better than PSP in execution time and the number of recursive calls; so it is faster in mining sequential patterns with time constraints from large databases.

V. Conclusion

We considered the problem of discovering sequential patterns by handling time constraints and comparing two algorithms (GTC and PSP). PSP and GTC algorithms adopted the general principles defined by GSP algorithm. But they are proposing a different data structure for storing candidates and frequent sequences. These differences make these algorithms

TABLE I
THE MAIN DIFFERENCES BETWEEN GTC AND PSP ALGORITHMS

	GTC algorithm	**PSP algorithm**
Structure	graph	prefix-tree
Execution time (Based on windowSize)	(1-40) sec	(1-120) sec
Number of recursive calls (Based on windowSize)	(1-9) million	(1-30) million
Candidate generation	No	Yes
Space-search	Backtracking	Backtracking

faster in mining sequential patterns with time constraints. Because they are avoiding useless and costly operations when verifying candidates. In PSP and GTC, time constraints are handled in the earlier stage of the algorithm to provide significant benefits.

After comparing the results of GTC and PSP algorithms; based on their experiments it is clear that the GTC algorithm is more efficient than PSP algorithm. To take the advantage of the behavior of the algorithm in the first scans on the database, we designed a new algorithm called PG-Hybrid using the PSP approach for the two first passes on the database and GTC for the following scans. The experiments showed that this approach is very efficient even for short, frequent sequences.

REFERENCES

[1] F. Masseglia, P. Poncelet, M. Teisseire. "Efficient mining of sequential patterns with time constraints: Reducing the combinations". Expert Systems with Applications, Volume 36, pages 2677-2690, 2009.
[2] F. Cathala, F. Masseglia, P. Poncelet. "The PSP Approach for Mining Sequential Patterns". ACM, pagesv176-184, 1998.
[3] Th. Rincy.N, Y. Pandey. "Performance evaluation on state of the art sequential pattern mining algorithms". International Journal of Computer Applications (0975-8887), Volume 65, N0. 14, March 2013.
[4] M. Lin, S. Hseueh, C. Chang. "Mining closed sequential patterns with time constraints". International Journal Of Information Science And Engineering Volume 24, pages 33-46, 2008.
[5] M. Morzy, T. Zakrzewicz, "Efficient constraint-based sequential pattern mining using dataset filtering techniques". In Proceedings of the Baltic conference, BalticDB IS, pages 213-224, 2002.
[6] M. J. Zaki, "SPADE: An efficient algorithm for mining frequent sequences". Machine Learning Journal, Volume 42, NO. 1, pages 31-60, 2001.
[7] R. Srikant and R. Agrawal. "Mining Sequential Patterns: Generalizations and Performance Improvements". In Proc. of the EDBT'96, Avignon, France, Sept 1996.
[8] U. M. Fayad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. "Advances in Knowledge Discovery and Data Mining". AAAI Press, 1996.
[9] J. Wijsen. "Condensed Representation of Database Repairs for Consistent Query Answering". In International Conference on Database Theory (ICDT), Springer-Verlag, LNCS 2572, pages 378-393, 2003.
[10] P. C. Kanellakis. "Elements of Relational Database Theory". In Jan van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, chapter 17, Elsevier/MIT Press, pages 1073-1158, 1990.
[11] J. Fry, G. Xian, S. Jin, J. Dewitz, C. Homer, L. Yang, C. Barnes, N. Herold, J. Wickham. "Completion of the 2006 National Land Cover Database for the conterminous United States", Photogrammetric Engineering Remote Sensing, Volume 77, NO. 9, pages 858-863, 2011.
[12] B. C. Ooi, C. Yu, K. L. Tan, and H. V. Jagadish. "Indexing the distance: an efficient method to knn processing". In Procdf the Int. Conf. on Very Large Data Bases, 2001.
[13] K. V. Ravi Kanth, D. Agrawal, Amr El Abbadi, and Ambuj K. Singh. "Dimensionality reduction for similarity searching in dynamic databases". In Proc. A CM SIGMOD Int. Conf. on Management of Data, 1998.
[14] J. Griss, Y. Perez-Riverol, H. Hermjakob, J. A. Vizcaino. "Identifying novel biomarkers through data mining-A realistic scenario". Proteomics Clin. Appl., Volume 9, pages 437-443, 2015.
[15] N. Roussopoulos, S. Kelley, and F. Vincent. "Nearest neighbor queries". In Proc. A CM SIGMOD Int. Conf. on Management of Data, pages 71-79, May 1995.
[16] H. Samet. "Recent developments in linear quadtree-based geographic information systems". Image and Vision Computing, Volume 5, NO. 3, pages 187-197, Aug. 1987.
[17] T. Sellis, N. Roussopoulos, and C. Faloutsos. "The r+-tree: A dynamic index for multi-dimensional objects". Procdf the Int. Conf. on Very Large Data Bases, Volume 13, pages 507-518, 1988.
[18] Y. Theodoridis and T. K. Sellis. "Optimization issues in r-tree construction". In Geographic Information Systems (IGIS), pages 270-273, 1994.
[19] F. Wang. "Relational-linear quadtree approach for two-dimensional spatial representation and manipulation". IEEE Trans. on Knowledge and Data Engineering, Volume 3, NO. 1, pages 118-122, Mar. 1991.
[20] J. Chomicki, J. Marcinkowski, and S. Staworko, "Computing consistent query answers using conflict hypergraphs," in CIKM, D. Grossman, L. Gravano, C. Zhai, O. Herzog, and D. A. Evans, Eds. ACM, pages 417-426, 2004.