Constraint-Based Mining of Sequential Patterns over Datasets with Consecutive Repetitions

Marion Leleu [1,2], Christophe Rigotti[1], Jean-François Boulicaut[1], and Guillaume Euvrard[2]

[1]LIRIS CNRS FRE 2672
Bâtiment Blaise Pascal, INSA Lyon, 69621 Villeurbanne Cedex, France
{crigotti, jfboulic}@lisisun1.insa-lyon.fr

[2] Direction de la Stratégie - Informatique CDC 113 rue Jean-Marin Naudin F-92220 Bagneux, France
{marion.leleu, guillaume.euvrard}@caissedesdepots.fr

Abstract. Constraint-based mining of sequential patterns is an active research area motivated by many application domains. In practice, the real sequence datasets can present consecutive repetitions of symbols (e.g., DNA sequences, discretized stock market data) that can lead to a very important consumption of resources during the extraction of patterns that can turn even efficient algorithms to become unusable. We propose a constraint-based mining algorithm using an approach that enables to compact these consecutive repetitions, reducing drastically the amount of data to process and speeding-up the extraction time. The technique introduced in this paper allows to retain the advantages of existing state-of-the-art algorithms based on the notion of occurrence lists, while permitting to extend their application fields to datasets containing consecutive repetitions. We analyze the benefits obtained using synthetic datasets, and show that the approach is of practical interest on real datasets.

Keywords: constraint-based mining, sequential pattern, generalized occurrence

1 Introduction

Sequential pattern mining has been introduced in 1995 [1]. It concerns pattern discovery (e.g., regularities) from ordered data, typically sequence databases. It has many applications, e.g., customer purchase analysis, Web Usage Mining, DNA sequence analysis. Looking for efficient algorithms has received a lot of attention (e.g., [8,11,9,5,10,12,14,13]). Each of these algorithms has its own pros and cons. Their efficiency depends on the characteristics of the data and on the

* This research is partially funded by the European Commission IST Programme - Future and Emergent Technologies, cInQ project (IST-2000-26469).

kind of user-defined selection criteria, i.e., the constraints that must be satisfied by the extracted patterns. Several available algorithms are based on the so-called *occurrence lists*, i.e., lists that contain the location of the patterns in the data. This technique has been proved very useful for frequent pattern extraction (e.g., [8,12,14,3,13]).

Independently, the use of user-defined constraints to reduce the search space during sequential pattern extraction has been developed (e.g., [11,9,4,2]). Indeed, it has also been integrated in the occurrence list approach in the *cSpade* algorithm [13], resulting in one of the most efficient algorithms proposed for constraint-based mining of sequential patterns.

We have two main application domains for which we need efficient sequential pattern algorithms: financial data (stock market data) analysis for CDC (a major financial company in France) and DNA sequence database analysis. When considering the *cSpade* approach on these data, we understood that the benefits of the use of occurrence lists are lost when mining sequences containing consecutive repetitions of symbols. It comes from an explosion of the number of occurrences due to the repetition of the symbols. We recently proposed to handle efficiently the repetitions in the occurrence lists [7] when considering only a minimal frequency constraint. In this paper, we present how to generalize the notion of occurrence to perform efficient constraint-based mining on collections of sequences that contain repetitions. From a practical point of view, this leads to a technique that retains the advantages of the *cSpade* approach, while being able to address efficiently a broader scope of applications. The key idea is to use a single generalized occurrence to represent several occurrences while keeping enough information for the mining process.

This paper is organized as follows. Section 2 recalls the constraint-based sequential pattern mining problem and gives an abstract formulation of an algorithm for sequential pattern mining using occurrence lists. The notion of generalized occurrence is introduced in Section 3, and the corresponding modifications of the mining algorithm is presented. The practical impact of the use of generalized occurrences is demonstrated by means of experiments in Section 4. We conclude in Section 5.

2 Problem Statement and Abstract Algorithm

2.1 Constrained Sequential Pattern

The problem is to mine all frequent sequential patterns, verifying some user-defined constraints, that can be found in a sequence database. The constraints considered in this paper are the so-called *minimum* and *maximum gap* constraints, that enable to specify the minimum or maximum time interval between the occurrences of two events inside a pattern. Another similar constraint considered is the *time window* constraint, that enables to limit the maximum time between the first event and the last event of a pattern. Basically, the problem can be presented as follows: Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m distinct items. An

event (also called *itemset*) of size l is a non empty set of l items from $I : (i_1 i_2 ... i_l)$. A *sequence* α of *length* L is an ordered list of L events $\alpha_1, \ldots, \alpha_L$, denoted as $\alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow \alpha_L$. A database is composed of sequences, where each sequence has a unique sequence identifier (sid) and each event of each sequence has a temporal event identifier (eid) called timestamp. For a sequence in the database, each eid associated to an event is unique and if an event e_i precedes event e_j in a sequence, then the eid of e_j must be strictly greater than the eid of e_i. A *sequential pattern* (or *pattern*) is a sequence. Due to the lack of space, we considered only single-item events in patterns, that is patterns composed of events of size 1. The extension to pattern composed of events of size greater than 1 is straightforward and can be found in an extended version of the paper [6].

We are interested in the so-called constrained sequential patterns defined as follows. A sequence $s_a = \alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$ is called a *subsequence* of another sequence $s_b = \alpha'_1 \rightarrow \alpha'_2 \rightarrow \ldots \rightarrow \alpha'_m$ if and only if there exists integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $\alpha_1 \subseteq \alpha'_{i_1}$, $\alpha_2 \subseteq \alpha'_{i_2}$, ..., $\alpha_n \subseteq \alpha'_{i_n}$. Let $supMin$ be a positive integer called *absolute support threshold*, a pattern p verifies the minimum frequency constraint in a database D if p is a subsequence of at least $supMin$ sequences of D. In this paper, we also use interchangeably the *relative support threshold* expressed in the percentage of the number of sequences of D. Let $gapMin$ be the fixed value of the *minimum gap* constraint. A pattern $p = \alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$ verifies the minimum gap constraint if and only if, for all α_i, $i = 1 \ldots n-1$, $eid(\alpha_{i+1}) - eid(\alpha_i) \geq gapMin$. Similarly, let $gapMax$ be the fixed value of the *maximum gap* constraint. Pattern p verifies the maximum gap constraint if and only if, for all α_i, $i = 1 \ldots n-1$, $eid(\alpha_{i+1}) - eid(\alpha_i) \leq gapMax$. Now, let $winMax$ be the fixed value of the *time window* constraint. Pattern p verifies this constraint, if and only if $eid(\alpha_n) - eid(\alpha_1) \leq winMax$.

2.2 Abstract Mining Algorithm

We present in this section an abstract algorithm corresponding to the general principle used in algorithms based on the use of occurrence lists for mining sequential patterns (e.g., [8,12,14,3,13]). The algorithm repeats two operations: a generation of candidate patterns and a support counting step. Let us introduce some needed concepts. A pattern with k items is called a *k-pattern*. A *prefix* of a k-pattern z is a subpattern of z constituted by the $k-1$ first items of z and its *suffix* corresponds to its last item. We extend the notion of *prefix* and *suffix* to occurrence. Let $y = e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{k-1} \rightarrow e_k$ be an occurrence of a k-pattern z, then $prefix(y) = e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{k-1}$ and $suffix(y) = e_k$.

The algorithm uses two frequent k-patterns z_1 and z_2 having the same $(k-1)$-pattern as prefix to generate a $(k+1)$-pattern z. This operation is denoted as $merge(z_1, z_2)$ and generates a single k-pattern: $z = z_1 \rightarrow suffix(z_2)$. The support counting for the newly generated pattern is not made by scanning the whole database. Instead, the algorithm has stored in specific lists, called $occLists$, the positions where z_1 and z_2 occur in the database. It then uses these two lists denoted $occList(z_1)$ and $occList(z_2)$ to determine where z occurs. Then $occList(z)$ allows to compute directly the support of z, by counting the number of distinct

sids present in this list. The computation of $occList(z)$ is a kind of *join* and is denoted $join(z_1, z_2)$. The abstract algorithm is presented as Algorithm 1.

Algorithm 1 (Abstract Mining Algorithm)

Input: *a database of sequences and a support threshold.*
Output: *the frequent sequential patterns contained in the database.*

```
Use the database to compute:
  - F_1 the set of all frequent items
  - occList(z) for all element z of F_1
let i := 1
while F_i ≠ ∅ do
  let F_{i+1} := ∅
  for all z_1 ∈ F_i do
    for all z_2 ∈ F_i do
      if z_1 and z_2 have the same prefix then
        let z := merge(z_1, z_2)
        let occList(z) := join(occList(z_1), occList(z_2))
        Use occList(z) to determine if z is frequent
        if z is frequent then
          F_{i+1} := F_{i+1} ∪ {z}
        fi
      fi
    od
  od
  i := i + 1
od
output ⋃_{1≤j<i} F_j
```

Fig. 1. Abstract mining algorithm using occurrence lists.

3 Generalized Occurrences and *GoSpec* Algorithm

3.1 Constrained Generalized Occurrences

The structure of a constrained generalized occurrence list is designed to reduce the size of the occurrence lists by representing several occurrences with a single more general one. In case of data presenting consecutive repetitions of items, this leads to an important gain in term of memory space used, and since the lists proceeded by the *join* operation are shorter, it results also in the reduction of the overall execution time.

For example, let us consider the following toy database containing three sequences. In these sequences the events are located at consecutive timestamps (i.e., 1,2,3, ...) and each sequence begin at timestamp 1.
Sequence 1:
$\{A\},\{A\},\{A\},\{A\},\{A\},\{B\},\{B\},\{B\},\{B,C\},\{B,C\},\{B,C\},\{B,C\},$
$\{B\},\{B\},\{B\}$
Sequence 2:
$\{B\},\{A,B\},\{A,B\},\{B\},\{B,C\},\{B,C\},\{B,C\},\{B,C\},\{B,C\},\{B,C\},$
$\{C\},\{C\},\{C\},\{C\}$
Sequence 3:
$\{\},\{A\},\{\},\{B\},\{B\},\{B\},\{B\},\{B,C\},\{B,C\},\{C\},\{C\},\{C\},\{C\},\{C\}$

A classical representation of occurrence lists like the one used by *cSpade* [13] is depicted in Figure 2, in the left tables of each three areas. These tables represent the occurrence lists of *cSpade* for patterns A, B, C, A → B, A → C and A → B → C, with supMin = 2, gapMin = 2, gapMax = 5 and winMax = 10. In the tables, the column *sid* corresponds to the identifier of the sequence in which the pattern occurs, *eid* corresponds to the timestamp of the last event of this occurrence, and *diff* corresponds to the difference between the timestamps of the first and the last event of the occurrence (used by *cSpade* to check the *time window* constraint).

We propose a notion of *constrained generalized occurrence* (generalized occurrence for short) to compact such consecutive occurrences. This notion is straightforward for pattern of size 1, but not so trivial for longer patterns since it has to enable the handling of the various constraints. For a pattern z, the form of a generalized occurrence is $\langle sid, tBeg, [min, max], gmax\rangle$, and contains:

- An identifier *sid* that corresponds to identifier of a sequence where pattern z occurs.
- A timestamp *tBeg* that corresponds to the timestamp of an occurrence of the first event of the pattern z (the detailed construction of *tBeg* will be given in Algorithm 2).
- An interval [*min*,*max*] corresponding to *eids* of consecutive occurrences of the last event of pattern z.
- A value *gmax* that indicates the timestamp of the last occurrence of the last event of pattern z respecting the $gapMax$ constraint. If no such occurrence exists then *gmax* is set to -1.

Examples of generalized occurrences for the toy database are given in Figure 2, in the right tables of each three areas. In the case of pattern B, it is possible to reduce its 10 consecutive occurrences in the first sequence to a single generalized occurrence $\langle 1, 6, [6, 15], 15\rangle$, where the interval [6,15] compacts all 10 *eids*. It should be noticed that for patterns of size 1 the fields *tBeg* and *gmax* are useless. However this is not the case for longer patterns. For example, let us consider the last generalized occurrence of the constrained generalized occurrence list of pattern A → B. This generalized occurrence is $\langle 3, 2, [4, 9], 7\rangle$, indicating that it appears in sequence 3 and starts at timestamp 2. The interval

[4,9] means that it represents several occurrences ending from 4 to 9. The *gmax* value of 7 notifies that occurrences ending from 4 to 7 satisfy the *maxGap* constraint, while for occurrences ending strictly after timestamp 7 only the prefix of the occurrence satisfies *maxGap*.

In the case of a generalized occurrence that does not represent any occurrence that satisfy the *maxGap* constraint for all its events, but that represents only occurrences satisfying this constraint up to this its last event, then the *gmax* value is set to -1 (as for example in the generalized occurrence $\langle 3, 2, [8, 12], -1\rangle$ of pattern A $\rightarrow$ C in Figure 2).

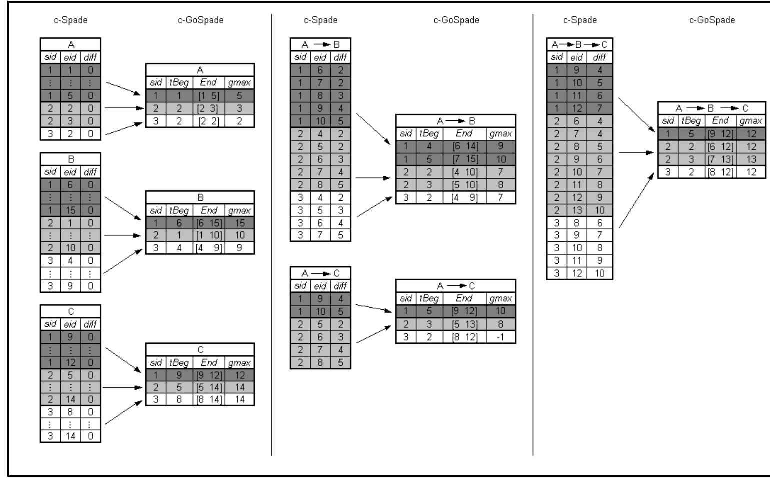

Fig. 2. Occurrence lists vs. Generalized occurrence lists for patterns A, B, C, A $\rightarrow$ B, A $\rightarrow$ C and A $\rightarrow$ B $\rightarrow$ C, with supMin = 2, gapMin = 2, gapMax = 5 and winMax = 10.

3.2 Dedicated Join Algorithm

The *GoSpec* Algorithm is an instance of the abstract algorithm 1 using a join designed for the generalized occurrence lists.

The join process is called when the merge operation has been done. It computes the constrained generalized occurrence list of a candidate pattern z, from the *occLists* of two generator patterns z_1 and z_2 having the same prefix.

Two different procedures are called depending on the level of the extraction process, $JoinLevel_2$ (Algorithm 4) and *Join* (Algorithm 3). The first one is a specific algorithm dedicated to the particular case of a 2-pattern candidate and the second one to the general case of a k-pattern candidate with $k > 2$. These two algorithms use a common function, *LocalJoin* (Algorithm 2), that computes a generalized occurrence $v = \langle sid, tBeg, [min, max], gmax\rangle$ of z from a single generalized occurrence of z_1 and a single generalized occurrence of z_2.

Algorithm 2 (*LocalJoin*)
Input: *Two generalized occurrences*
$\langle sid_1, tBeg_1, [min_1, max_1], gmax_1 \rangle$
and $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2 \rangle$
Output: $\langle v, add \rangle$, *where* $v = \langle sid, tBeg, [min, max], gmax \rangle$ *and* add *is a boolean value that is false if v cannot be created.*

```
1. let add := false
2. let v := null
3. if (min_1 + gapMin ≤ max_2) and (tBeg_1 + winMax ≥ min_2)
         and (min_1 ≤ gmax_1) then
4.   if (sid_1 = sid_2) then
5.     let sid := sid_1
6.     let tBeg := tBeg_1
7.     find min the minimum element x of [min_2, max_2]
           such that x ≥ min_1 + gapMin
8.     find max the maximum element x of [min_2, max_2]
           such that x ≤ tBeg_1 + winMax
9.     find gmax the maximum element x of [min_2, max_2]
           such that x ≤ gmax_1 + gapMax
10.  fi
11.  if (min and max exist) and (min ≤ max) then
12.    if (gmax not exists) then let gmax := −1
13.    else if (gmax > max) then
14.      let gmax := max fi
15.    fi
16.    let v := ⟨sid, tBeg, [min, max], gmax⟩
17.    let add := true
18.  fi
19. fi
20. output ⟨v, add⟩
```

Algorithm 3 (*Join*)
Input: *occList(z_1) and occList(z_2), generalized occurrence lists of two patterns that share a same prefix.*
Used subprograms: *Algorithm 2*
Output: *a new occList*

```
Initialize GoIdList to the empty list.
1. for all occ_1 ∈ GoIdList(z_1) do
2.   for all occ_2 ∈ GoIdList(z_2) do
3.       let ⟨v, add⟩ := LocalTemporalJoin(occ_1, occ_2)
4.       if add then
5.         Insert v in occList
6.       fi
7.   od
8. od
9. output occList
```

Fig. 3. *LocalJoin* and *Join* algorithms.

The $LocalJoin$(Algorithm 2), first verifies that the input generalized occurrences satisfy necessary conditions to be joined, performing the tests of line 3 and that the two generalized occurrences are from a same sequence, that is $sid_1 = sid_2$ (line 4). One line 3, the first comparison verifies that there exists at least one suffix of an instance of $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ that follows the first suffix of an instance of $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ and that satisfies the $gapMin$ constraint. The second comparison checks that there exists at least one suffix of an instance of $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ that satisfies the $winMax$ constraint wrt. $tBeg_1$. The last comparison ensures that $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ has at least one instance that satisfies the $gapMax$ constraint.

Lines 5 to 9 generate a new generalized occurrence. min is the timestamp of the earliest suffix of an instance of $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ that follows the earliest suffix of an instance of $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ and that verifies the *minimum gap* constraint. In a same way, max is the timestamp of the latest suffix of an instance of $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ that verifies the *time window* constraint wrt $tBeg_1$. $gmax$ indicates the timestamp of the latest suffix of an instance of $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ that can form an occurrence of v that verifies $gapMax$.

This $LocalJoin$ algorithm is called by $Join$ (Algorithm 3) that generates a new *occList* from the *occLists* of two generator patterns z_1 and z_2. The Algorithm 3 iterates on the elements of $occList(z_1)$ and $occList(z_2)$. For each pair (occ_1,occ_2) a new constrained generalized occurrence is generated when possible using $LocalJoin$. Algorithm 3 is the general join operation used for k-patterns when $k > 2$. A dedicated join is needed to generate the occurrence lists of 2-patterns (i.e., z_1 and z_2 contain a single item. It is called $JoinLevel_2$ and is presented as Algorithm 4. Contrarly to the general $Join$, $JoinLevel_2$ performs several calls to the $LocalJoin$ procedure. Indeed, the instances of the generalized occurrence $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ must be proceeded separately because they correspond, in the data, to different starting timestamps of the 1-pattern z_1. Thus, several calls are made on all generalized occurrences $\langle sid_1, p, [p, p], p\rangle$ with p varying between the values min_1 and max_1.

Proofs of the correctness of the representation using generalized occurrences (and the corresponding join process) can be found in [6].

4 Experimental Results

In this section, we present experimental results and compare the behaviors of *GoSpec* and of *cSpade* [13] (one of the most efficient algorithm proposed in the literature and based on occurrence lists).

Both algorithms have been implemented using Microsoft Visual C++ 6.0, with the same kind of low level optimization to allow a fair comparison. All experiments have been performed on a PC with 196 MB of memory and a 500 MHz Pentium III processor under Microsoft Windows 2000.

```
Algorithm 4 (JoinLevel_2)
Input: occList(z_1), occList(z_2)
Used subprograms: Algorithm 2
Output: a new occList

Initialize occList to the empty list.
1.for all ⟨sid_1, tBeg_1, [min_1, max_1], gmax_1⟩ ∈ occList(z_1) do
2.  for all ⟨sid_2, tBeg_2, [min_2, max_2], gmax_2⟩ ∈ occList(z_2) do
3.    for all p ∈ [min_1 , max_1] do
4.      let ⟨v, add⟩ := LocalTemporalJoin(⟨sid_1, p, [p, max_1], p⟩,
               ⟨sid_2, tBeg_2, [min_2, max_2], gmax_2⟩)
5.      if add then
6.        Insert v in occList
7.      fi
8.    od
9.  od
10.od
11.output occList
```

Fig. 4. $JoinLevel_2$ algorithm.

4.1 Experiments on synthetic datasets

The synthetic dataset has been generated using the Dataquest generator of IBM [1] and the following parameters: C10-T2.5-S4-I1.25-D1K over an alphabet of 100 items (called $set1$). It contains 1000 sequences with an average size of 10 events per sequences (see [1] for more details on the generator parameters). In this dataset, the time interval between two time stamps is 1, and there is one event per time stamp.

In order to have datasets presenting parameterized consecutive repetitions on certain items, we performed a post-processing on $set1$ to add such repetitions. Each item founded in an event of a sequence has a probability fixed to 10% to be repeated. When an item is repeated, we simply duplicated it in the next i consecutive events. If the end of the sequence is reached during the duplication process the sequence is not extended (no new event is created) and thus, the current item is not completely duplicated. We denote $set1_r\{i\}$ the dataset obtained with a repetition parameter of value i. For the sake of uniformity, $set1$ is denoted $set1_r0$. The post-processing on $set1_r0$ leads to the creation of 5 new datasets $set1_r1$,..., $set1_r5$.

The three first graphs (top-left, top-right and middle-left) of Figure 5 show the results of the extractions performed on datasets $set1_r1$, $set1_r2$, ..., $set1_r5$ with the following constraints: a support threshold fixed to 2.5%, a window time limited to 6, a minimum gap fixed to 2 and a maximum gap fixed to 4.

The top-left graph (Figure 5) gives the size of the *cSpade* and *GoSpec* occurrences lists (in number of elements) for extraction performed on files $set1_r1$, ..., $set5_r5$. As expected, the total number of occurrences used by *cSpade* is greater

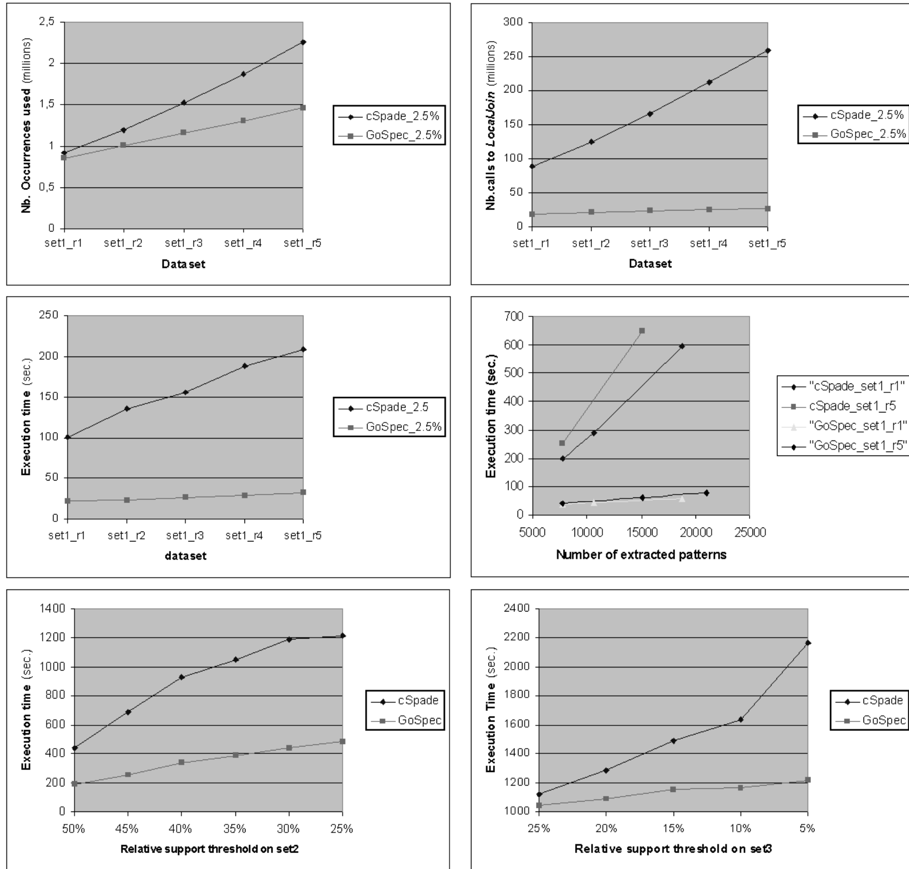

Fig. 5. Experiments using *GoSpec* and *cSpade*.

than the number of constrained generalized occurrences used by *GoSpec*, and this reduction increases with the number of repetitions. The top-right graph shows that this reduction has a direct impact on the join costs (in term of number of calls to *LocalJoin*), that results on an important reduction of the total execution time of the extractions as shown in the middle-left graph of Figure 5.

The middle-right graph of Figure 5 completes these results with the extraction times on datasets $set1_r0$ and $set1_r5$. It shows that the execution time to find a given number of patterns remains quite the same in presence of repetitions for *GoSpec*.

4.2 Experiments on real datasets

The first real dataset is a financial dataset provided by the CDC financial company (Caisse des Dépots et Consignations) and contains the variations of stock

prices over one year. The discretized data results in a set (called $set2$) of 2830 sequences with an average length of 15 events per sequence. These sequences have been built from an alphabet of 17 items. The extractions have been performed using the extended version of the algorithm ([6]), that is without any limitation on the number of item per event composing the generated patterns. The following constraints have been used: winMax = 10, maxGap = 4 and minGap = 2. The bottom-left graph of Figure 5 represents the total execution time of both *cSpade* and *GoSpec* for minimum support thresholds varying from 25% to 50% and shows that *GoSpec* offers a significant gain wrt. *cSpade*.

The second real dataset corresponds to a dataset of DNA sequences called $set3$. It contains 1778 sequences with an average length of 102 events composed by only one item per event over the nucleic alphabet {A,T,G,C}. The extractions have been performed using a window time constraint sets to 6, a maximum gap constraint of 3, no minimum gap constraint, and a minimum support threshold varying from 5% to 25%. The bottom-left graph of Figure 5 illustrates the total execution time used by the extractions and shows the advantages of *GoSpec* in practice on this second kind of data.

5 Conclusion

In this paper we presented an algorithm that enables to manage efficiently the constraint-based mining task when the sequential databases contain consecutive repetitions of their items. Such a situation can appear in several domains (e.g., discretized quantitative time series and DNA sequences). This can cause an explosion of the number of pattern occurrences and thus to an important loss of efficiency for algorithms based on an occurrence list approach (e.g., [8,12,14,3,13], while this algorithm family has shown its interest in many situations (e.g., low support mining and active constraint handling)). The algorithm presented in this paper, extends this family to tackle with these domains. It is based on the notion of constrained generalized occurrences, that have the particularity to compact several consecutive occurrences of patterns while keeping enough information for a constraint-based mining process. We showed by means of experiments , that the gain in term of memory space and execution time is important and that it increases with the number of consecutive repetitions contained in the input sequences.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995. IEEE Computer Society.
2. H. Albert-Lorincz and J.-F. Boulicaut. Mining frequent sequential patterns under regular expressions: a highly adaptive strategy for pushing constraints. In *Proceedings of the Third SIAM International Conference on Data Mining SDM 2003*, San Francisco, USA, May 2003.

3. J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using bitmap representation. In *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 2002.
4. M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proc. of the 25th International Conference on Very Large Databases (VLDB'99)*, pages 223–234, Edinburgh, United Kingdom, September 1999.
5. J. Han, J. Pei, B. Han Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, pages 355–359, Boston, MA, USA, August 2000.
6. M. Leleu, C. Rigotti, J.-F. Boulicaut, and G. Euvrard. Constrained-based mining of sequential patterns over datasets with consecutive repetitions. Technical report, LIRIS, INSA Lyon, Bat. Blaise Pascal, 69621 Villeurbanne Cedex, France, 2003.
7. M. Leleu, C. Rigotti, J.-F. Boulicaut, and G. Euvrard. Go-spade: Mining sequential patterns over datasets with consecutive repetitions. In *Proc.2003 Int. Conf. Machine Learning and Data Mining (MLDM'03)*, Leipsig, Germany, July 2003.
8. H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–298, November 1997.
9. F. Masseglia, F. Cathalat, and P. Poncelet. The PSP approach for mining sequential patterns. In *Proc. of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery in Databases (PKDD'98)*, pages 176–184, Nantes, France, September 1998. Lecture Notes in Artificial Intelligence, Springer Verlag.
10. J. Pei, B. Han, B. Mortazavi-Asl, and H. Pinto. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. of the 17th International Conference on Data Engineering (ICDE'01)*, 2001.
11. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th International Conference on Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, September 1996.
12. M. Zaki. Efficient enumeration of frequent sequences. In *Proc. of the 7th International Conference on Information and Knowledge Management (CIKM'98)*, pages 68–75, November 1998.
13. M. Zaki. Sequence mining in categorical domains: incorporating constraints. In *Proc. of the 9th International Conference on Information and Knowledge Management (CIKM'00)*, pages 422–429, Washington, DC, USA, November 2000.
14. M. Zaki. Spade: an efficient algorithm for mining frequent sequences. *Machine Learning, Special issue on Unsupervised Learning*, 42(1/2):31–60, Jan/Feb 2001.