Methods for the Efficient Discovery of Large Item-Indexable Sequential Patterns

Rui Henriques[1,2], Cláudia Antunes[2], and Sara C. Madeira[1,2]

[1]KDBio, Inesc-ID, Instituto Superior Técnico, Universidade de Lisboa
[2]Dep. Computer Science and Engineering, IST, Universidade de Lisboa
{rmch,claudia.antunes,sara.madeira}@tecnico.ulisboa.pt

Abstract. An increasingly relevant set of tasks, such as the discovery of biclusters with order-preserving properties, can be mapped as a sequential pattern mining problem on data with item-indexable properties. An item-indexable database, typically observed in biomedical domains, does not allow item repetitions per sequence and is commonly dense. Although multiple methods have been proposed for the efficient discovery of sequential patterns, their performance rapidly degrades over item-indexable databases. The target tasks for these databases benefit from lengthy patterns and tolerate local mismatches. However, existing methods that consider noise relaxations to increase the average short length of sequential patterns scale poorly, aggravating the yet critical efficiency. In this work, we first propose a new sequential pattern mining method, IndexSpan, which is able to mine sequential patterns over item-indexable databases with heightened efficiency. Second, we propose a pattern-merging procedure, MergeIndexBic, to efficiently discover lengthy noise-tolerant sequential patterns. The superior performance of IndexSpan and MergeIndexBic against competitive alternatives is demonstrated on both synthetic and real datasets.

1 Introduction

Sequential pattern mining (SPM) has been proposed to deal efficiently with the discovery of frequent precedences and co-occurrences in itemset sequences. SPM methods can be applied to solve tasks centered on extracting order-preserving regularities, such as the discovery of flexible (bi)clusters [14]. These tasks commonly rely on a more restricted form of sequences, item-indexable sequences, which do not allow item repetitions per sequence. Illustrative examples of item-indexable databases include sequences derived from microarrays, molecular interactions, consumer ratings, ordered shoppings, tasks scheduling, among many others. However, these tasks are characterized by two major challenges. First, their hard nature, which is related with two factors: average high number of items per transaction and high data density. Second, order-preserving solutions are optimally described by lengthy noise-tolerant sequential patterns [5].

Although existing SPM approaches can be applied over item-indexable databases, they suffer from two problems. First, they show inefficiencies due to the commonly observed density levels and high average transaction length of these datasets, which leads to a combinatorial explosion of sequential patterns under low support thresholds [14]. Additionally, the few dedicated methods able to discover sequential patterns in item-indexable databases [13, 14] show significant memory overhead.

Second, the average length of sequential patterns is typically short. A common desirable property for the tasks formulated over these databases is the discovery of sequential patterns with a medium-to-large number of items. For instance, order-preserving patterns from ratings and biological data are only relevant above a minimum number of items. Such lengthy patterns can be discovered under very low support thresholds, aggravating the yet hard computational complexity, or by assuming local noise (violation of ordering constraints for a few transactions). However, some of existing SPM extensions to deal with noise relaxations have been tuned for different settings [6], while others can deteriorate the yet critical SPM efficiency [25]. Additionally, methods to discover colossal patterns from smaller itemsets [30] have not been extended for the SPM task.

This work proposes a new method for the efficient retrieval of lengthy order-preserving regularities based on sequential patterns discovered over item-indexable sequences. This is performed in two steps. First, we propose a new method, IndexSpan, that uses efficient data structures to keep track of the position of items per sequence and relies on fast database projections based on the relative order of items. Pruning techniques are available when the user has only interest in sequential patterns above a minimum length. Second, we propose an efficient method, MergeIndexBic, to guarantee the relevance of order-preserving regularities by deriving medium-to-large sequential patterns from multiple short sequential patterns. MergeIndexBic uses an error threshold based on the percentage of shared sequences and items among sets of sequential patterns. This is accomplished by mapping this problem in one of two tasks: discovery of maximal circuits in graphs or multi-support frequent itemset mining.

The paper is structured as follows. *Section 2* introduces and motivates the task of mining sequential patterns over item-indexable databases, and covers existing contributions in the field. *Section 3* describes the proposed solution based on the IndexSpan and MergeIndexBic methods. *Section 4* assesses the performance of IndexSpan on both real and synthetic datasets against SPM and dedicated algorithms. The performance of MergeIndexBic against default alternatives is also validated. Finally, the implications of this work are synthesized.

2 Background

Let an item be an element from an ordered set $\mathcal{L}$. An *itemset* I is a set of non-repeated items, $I \subseteq \mathcal{L}$. A *sequence* s is an ordered set of itemsets. A sequence a=<$a_1 ... a_n$> is a *subsequence* of b=<$b_1 ... b_m$> ($a \subseteq b$), if $\exists_{1 \leq i_1 < .. < i_n \leq m}$: $a_1 \subseteq b_{i_1}$,..,$a_n \subseteq b_{i_n}$. The illustrative sequence s_1=<$\{a\}, \{be\}$>=$a(be)$ is contained in s_2=$(ad)c(bce)$. A *sequence database* is a set of sequences $D = \{s_1, .., s_n\}$.

The **coverage** Φ_s of a sequence s w.r.t. to a set of sequences D, is the set of all sequences in D with s as subsequence: $\Phi_s = \{s' \in D \mid s \subseteq s'\}$. The **support** of a sequence s in D, denoted sup_s, is its coverage size $|\Phi_s|$. Illustrating, consider the sequence database D=$\{s_1$=$(bc)a(abc)d, s_2$=$cad(acd), s_3$=$a(ac)c\}$. For this database, we have $|\mathcal{L}|$=4, $\Phi_{\{a(ac)\}}$=$\{s_1, s_2, s_3\}$, and $sup_{\{a(ac)\}}$=3.

Given a set of sequences D and some user-specified minimum support threshold θ, a sequence $s \in D$ is *frequent* when is subsequence of at least θ sequences.

The **sequential pattern mining** (SPM) problem consists of computing the set of frequent sequences, $\{s \mid sup_s \geq \theta\}$.

The set of maximal frequent sequences for the illustrative sequence database, D=$\{(bc)a(abc)d, cad(acd), a(ac)c\}$, under a minimum support θ=3 is $\{a(ac), cc\}$.

Let an item-indexable sequence be a sequence without repeated items. An item-indexable sequence database is a set of item-indexable sequences.

Let $|I|$ be the length of an itemset, and $|s|$ be the length of a sequence, $\Sigma_i\ |s^i|$. Given a set of item-indexable sequences D, a minimum support threshold θ, and a minimum sequence length δ. The task of **SPM over item-indexable sequences**, or simply item-indexable SPM, consists of computing:

$$\{s \mid sup_s \geq \theta \wedge\ |s| \geq \delta\}$$

This formalization allows the definition of new methods prone to seize the properties of item-indexable sequences. Understandably, the resulting sequential patterns, referred as item-indexable sequential patterns, preserve the consistency of item ordering constraints since they do not allow for item duplicates.

2.1 Applications

From the large set of applications of item-indexable SPM[1], one prominent task is order-preserving biclustering, a form of local clustering based on frequent ordering constraints [13, 5]. Order-preserving biclustering is commonly applied over biological domains for the analysis of gene expression data, networks and genomic structural variations. Illustrating, finding subsets of genes respecting orderings on the levels of expression across conditions is critical to study frequent variations, otherwise not discovered under the original levels of expression.

To compose an item-indexable database D from a real-value or discrete matrix, (X, Y), where X=$\{s_1, .., s_n\}$ is a set of rows and Y=$\{y_1, .., y_m\}$ is a set columns, the column indexes are linearly ordered for each transaction according to their values. Each transaction is seen as a sequence of items that correspond to column indexes. A bicluster, (I, J), a correlated subset of rows $I \subset X$ and columns $J \subset Y$, is order-preserving if the permutation of its columns J is strictly increasing across the I rows. A order-preserving bicluster can be derived from a frequent sequence s by mapping (I, J)=$(\Phi_s, \{s_i \mid i$=$1..|s|\})$. Fig.1 illustrates how order-preserving biclustering can be solved using SPM.

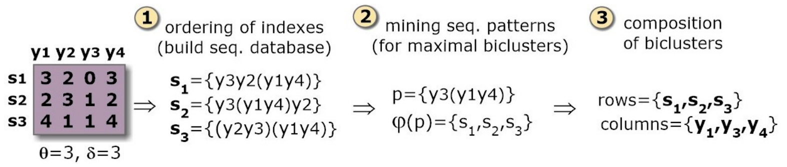

Fig.1: Mining order-preserving biclusters from item-indexable databases.

An increasingly important application of item-indexable SPM for recommendations based on user preferences, quality assessments and questionnaires [11]. Item-indexable sequences are derived from an ordering of ratings, such as ratings

[1] Detailed description of tasks availabe in http://web.ist.utl.pt/rmch/software/indexspan

of videos, hotels, shopped items, restaurants, among other products and experiences recorded in large-scale platforms (e.g. IMDb, booking.com, Amazon). Frequent precedences and co-occurrences disclose relevant priorities for different groups of users. Other applications include the discovery of order-preserving regularities in scheduling, planning, shopping, and traveling behavior [28, 9].

2.2 Related Work

Two major lines of research are considered. First, we review the general SPM methods and item-indexable dedicated methods and cover their major drawbacks. Second, we gather the potentialities and limitations of the available alternatives to compose lengthy (sequential) patterns.

Efficient SPM in item-indexable databases. Although general SPM methods are not optimized to deal with item-indexable specificities, they have been the largely adopted to solve these applications [14]. Since the SPM problem proposal [1], multiple extensions and applications have been proposed, ranging from scalable implementations to alternative pattern representations. Current SPM methods can be classified into three main categories: apriori-based, pattern-growth, and early-pruning [15]. Apriori-based algorithms [22], and vertical-based variations [27], rely on join procedures to generate candidate sequences in a breadth-first manner using multiple database scans. To overcome the computational complexity of maintaining the support count for each sequence generated, the use of bitmaps or direct comparison have been proposed [7, 3].

Pattern growth methods [17, 18, 3] avoid the costs from candidate generation by building a representation of the database and recursively traversing it to grow the frequent sequences. PrefixSpan [17], an efficient option, recursively constructs patterns by growing their prefix and by maintaining their corresponding postfix subsequences into projected databases. This guarantees a narrowed search space and avoids the generation of candidates since it only counts the frequency of local sequences. The major cost of PrefixSpan resides on database projections.

Early-pruning methods emerged more recently in the literature [26, 7, 20]. They adopt a sort of position induction to prune candidate sequences very early in the mining process and to avoid support counting as much as possible. These algorithms usually employ a table to track the last positions of each item in the sequence to evaluate whether the item can be appended to a given prefix.

The drawback of these SPM alternatives is that their performance does not scale for very low support thresholds, which are often required in item-indexable contexts to obtain medium-to-large sequential patterns. In fact, new methods can seize the item-indexable property, that guarantees that each item appears at most one time per item-indexable sequence, to minimize this problem.

Seizing this property, Liu and Wang [13, 14] proposed an alternative SPM method that constructs a compact tree structure, OPC-Tree, where sequences sharing the same prefix are gathered and recorded in the same branch. The discovery of frequent subsequences and the association of rows with frequent subsequences are performed simultaneously. However, the memory complexity of OPC-Tree is $\Theta(n \times m^2)$, where n is the number of records and m the average number of items per transaction. Although some pruning techniques can be applied in the OPC-Tree structure, their impact is not sufficient to turn this approach scalable for medium-to-large databases.

Discovery of Lengthy (Sequential) Patterns. The existing attempts for the discovery of lengthy patterns can be synthesized according to four major directions. First direction is to rely on efficient methods able to discover sequential patterns under very low support thresholds. There are two main classes of such methods. First class incorporates look-ahead heuristics, such as the ones used by MaxMiner [4], to avoid the traversal of every frequent sequence [12]. Second class generates patterns by reducing large candidates using the monotocity property of frequency (if an itemset is frequent, then its subsets are also frequent). Constraint programming methods can make good use of monotonic constraints to effectively reduce the search space [19]. However, under very low support thresholds, there are two structural problems. First, there is a high probability of discovering precedences and co-occurrences by chance due to the small set of supporting transactions. Second, the increase of length in the number of items comes at the cost of an heightened decrease in the number of supporting transactions. This does not support the final goal since the size of order-preserving regularities is defined by the length of both item and transaction sets (as previously illustrated in Fig.1). To overcome this drawback, the remaining directions allow for noisy patterns to increase their length without a significant decrease of support levels. These directions assume a tolerance of item mismatches observed for small subsets of the supporting transactions.

Second direction is to extend SPM to discover sequential patterns with local mismatches and gap-based relaxations [25, 2, 6, 29]. However, such extensions either: assume the presence of very long sequences for the creation of partitions, or increase the computational complexity of the original methods, limiting even more the discovery sequential patterns in useful time.

Third direction is to rely on approximative pattern mining under specific principle for composing lengthy patterns [8]. In particular, colossal pattern mining relies on the approximative fusion of smaller patterns [30]. However, these principles have been synthesized in the context of frequent itemset mining and, to our knowledge, have not yet been extended for sequential pattern mining.

Final direction is to view patterns as biclusters and rely on dedicated biclustering merging strategies [21]. The need for merging biclusters is based on the observation that when two biclusters share a significant area it is probable that they are part of a larger coherent bicluster. The simplest criterion for merging is to rely on the overlapping area (as a percentage of the larger bicluster). Nevertheless, the existing approaches require the computation of similarities between all pairs of biclusters. Understandably, this solution is impracticable for solutions characterized by a large number of biclusters (sequential patterns).

3 Solution

The proposed solution is defined by two major methods. First, IndexSpan method for the efficient discovery of sequential patterns over dense item-indexable databases. Second, MergeIndexBic method to consider relaxations that allow local ordering violations to compose larger and noise-tolerant sequential patterns.

3.1 IndexSpan: Boosting Item-Indexable SPM

To avoid the drawbacks of existing approaches, we propose the IndexSpan method, an extension of PrefixSpan to discover sequential patterns with heightened efficiency from item-indexable databases. Comparison of existing SPM algorithms [15] shows key heuristics to turn the SPM efficient: mechanisms to reduce the support counting; narrowing of the search space; optimally sized data structure representations of the sequence database; and early pruning of candidate sequences. Seizing these properties, IndexSpan guarantees a search space as small as possible and relies on a narrow search procedure, depth-first search.

IndexSpan extends PrefixSpan [17] in order to incorporate additional efficiency gains from three principles. First, IndexSpan relies on an easily indexable and compacted version of the original sequence database. Second, it uses faster and memory-efficient database projections. A projected database only maintains a list with the IDs of the active sequences. Finally, IndexSpan relies on early-pruning techniques. IndexSpan is described in Algorithm 1.

Algorithm 1: IndexSpan

```
Input: sequence database $D$, minimum support $\theta$, minimum sequence length $\delta$
Output: set of sequential patterns $S$
Note: $\alpha$ is a sequence, $D_\alpha$ is the $\alpha$-projected database
      ($D_\alpha$ simply maintains a reference to the current sequences)

mainMethod() begin
    foreach sequence $s$ in $D$ /*add array of item indexes per sequence*/ do
        foreach item $c$ do
            $s$.indexes[$c$] $\leftarrow$ position($s$,$c$);
    $\alpha$.items $\leftarrow \phi$; $\alpha$.trans $\leftarrow \phi$;
    indexSpan($\alpha$,$D$);

indexSpan($\alpha$,$D_\alpha$) begin
    foreach frequent item $c$ in $D_\alpha$ do
        $\beta$.items $\leftarrow \alpha.items \cup c$; //co-occurrence (c is added to the last $\alpha$ itemset)
        $\gamma$.items $\leftarrow \alpha.items \cdot c$; //$\alpha$ precedes c (c is inserted as a new itemset)
        //pruning and fast gathering of sup. transactions (for efficient data projection)
        foreach sequence $s$ in $D_\alpha$ do
            currentIndex $\leftarrow$ $s$.indexes[$c$];
            upperIndex $\leftarrow$ $s$.indexes[$\alpha_n$] /*$\alpha_n$ is the last item*/;
            if leftPositions(currentIndex)$\geq\delta$-$|\alpha|$ /*pruning*/ then
                if currentIndex > upperIndex then
                    $\gamma$.trans $\leftarrow \gamma$.trans $\cup$ $s$.ID;
                else
                    if currentIndex=upperIndex $\wedge$ $c>\alpha_n$ then $\beta$.trans $\leftarrow \beta$.trans$\cup s$.ID;
        if $sup_\beta(D_\alpha) \geq \theta$ then
            $S \leftarrow S \cup \{\beta\}$;
            $D_\beta \leftarrow$ fastProjection($\beta$,$D_\alpha$);
            indexSpan($\beta$,$D_\beta$);
        if $sup_\gamma(D_\alpha) \geq \theta$ then
            $S \leftarrow S \cup \{\gamma\}$;
            $D_\gamma \leftarrow$ fastProjection($\gamma$,$D_\alpha$);
            indexSpan($\gamma$,$D_\gamma$);

fastProjection($\beta$,$D_\alpha$) begin
    foreach sequence $s$ in $D_\alpha$ do
        currentIndex $\leftarrow$ $s$.indexes[$\beta_n$];
        upperIndex $\leftarrow$ $s$.indexes[$\beta_{n-1}$];
        if leftPositions(currentIndex)$\geq\delta$-$|\alpha|$ /*pruning*/ then
            if currentIndex > upperIndex then
                $D_\beta \leftarrow D_\beta \cup s$;
            else
                if currentIndex=upperIndex $\wedge$ $c > \alpha_n$ then $D_\beta \leftarrow D_\beta \cup s$;
    return $D_\beta$;
```

IndexSpan considers the three following structural adaptations over the PrefixSpan algorithm. First, it maintains a simple matrix in memory that maintains the index of each item per row. This matrix is constructed at the very beginning (*lines 2-5*) and the original database is removed. Additionally, for sparse databases, this matrix is replaced by a vector of hash tables to optimize memory usage. These data structures support position induction. The idea behind is simple: if an item's last/start position precedes the current prefix/postfix position, the item can no longer appear before/after the current prefix.

Second, a projected database can be constructed with heightened efficiency by avoiding the need to update and maintain postfixes. A projected database simply maintains the identifiers of the supporting sequences for a specific prefix.

To know if a sequence is still frequent when an item is added over a specific prefix, there is only the need to compare its index against the index of the previous item as well as their lexical order for the case where the index is the same (i.e. the new item co-occurs with the last items of the pattern). In this way, database projections, the most expensive step of PrefixSpan both in terms of time and memory, are handled with heightened efficiency. The proposed projection method is described in Algorithm 1, *lines 12-19* and *28-37*.

Finally, the input minimum number of items per sequential pattern, δ, can be used to prune the search as early as possible. If the number of items of the current prefix ($|\alpha|$) plus the items of a postfix s_α (computed based on the current and last index positions) is less than δ, then the sequence identifier related with the s_α postfix can be removed from the projected database since all the patterns supported by s will have a number of items below the inputted threshold.

For an optimal pruning, this assessment is performed before item indexes comparisons, which occurs in two distinct moments during the prefixSpan recursion (Algorithm 1 *lines 15* and *32*).

The efficiency gains from fast database projections and early pruning techniques, combine with the absence of memory overhead, turn IndexSpan highly attractive in comparison with the OPC-Tree peer method.

3.2 MergeIndexBic: Composing Large Item-Indexable Patterns

In real-world contexts, an ordering permutation observed among a set items can be violated for specific transactions due to the presence of noise. This can either result in a sequential pattern with a reduced set of transactions or items (if this violation turns the original sequence infrequent). In these scenarios, it is desirable to allow some of these violations. Four directions to accomplish this goal were covered in *Section 2.2*, with two directions being limited with regards to their outcome and the other two directions with regards to their levels of efficiency. In this section, we propose MergeIndexBic, which makes available two efficient methods to compose lengthy sequential patterns based on merging procedures.

Merging procedures have been applied over sets of biclusters by computing the similarities (overlapping degree) between all pairs of biclusters. Remind that a sequential pattern s can be viewed as a bicluster (I, J), by mapping the supporting transactions Φ_s as the I rows and the pattern items as the J columns. In this context, merging occurs when a set of biclusters (I_k, J_k) has an overlapping area above a minimum threshold, meaning that a new bicluster (I', J') is composed with $I' = \cup_k I_k$ and $J' = \cup_k J_k$. This new bicluster can be mapped back as a sequential pattern, for order-preserving tasks where biclustering is not the ultimate goal. This strategy is reliable as it considers both the set of shared items and the set of shared transactions to perform the merging step, leading to an effective identification and allowance of local ordering violations. A global view of this step is provided in Fig.2, where three larger biclusters are derived from subsets of biclusters satisfying the minimum overlapping constraint.

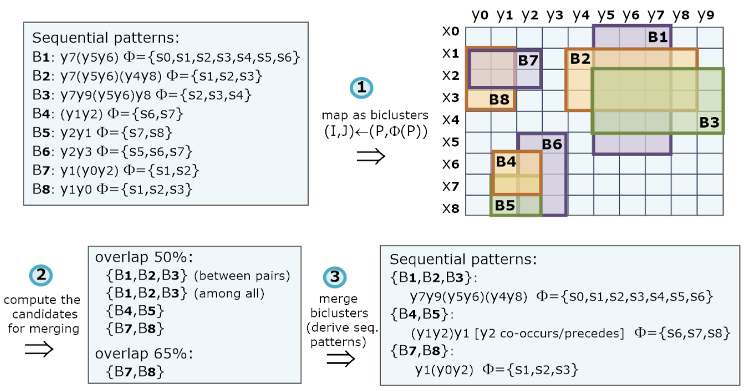
Sequential patterns:
B1: y7(y5y6) Φ={s0,s1,s2,s3,s4,s5,s6}
B2: y7(y5y6)(y4y8) Φ={s1,s2,s3}
B3: y7y9(y5y6)y8 Φ={s2,s3,s4}
B4: (y1y2) Φ={s6,s7}
B5: y2y1 Φ={s7,s8}
B6: y2y3 Φ={s5,s6,s7}
B7: y1(y0y2) Φ={s1,s2}
B8: y1y0 Φ={s1,s2,s3}
1
map as biclusters
(I,J)←(P,Φ(P))
y0 y1 y2 y3 y4 y5 y6 y7 y8 y9
x0 x1 x2 x3 x4 x5 x6 x7 x8
B1
B7
B2
B8
B3
B6
B4
B5
2
compute the candidates for merging
overlap 50%:
{B1,B2,B3} (between pairs)
{B1,B2,B3} (among all)
{B4,B5}
{B7,B8}
overlap 65%:
{B7,B8}
3
merge biclusters (derive seq. patterns)
Sequential patterns:
{B1,B2,B3}:
y7y9(y5y6)(y4y8) Φ={s0,s1,s2,s3,s4,s5,s6}
{B4,B5}:
(y1y2)y1 [y2 co-occurs/precedes] Φ={s6,s7,s8}
{B7,B8}:
y1(y0y2) Φ={s1,s2,s3}

Fig.2: Composing lengthy sequential patterns from smaller patterns by merging biclusters: considering the influence of both the set of items and the set of transactions.

In this illustration, three major steps are considering: *1)* mapping sequential patterns as sets of rows and columns; *2)* discovering candidates for merging; and *3)* recovering the new sequential patterns. In particular, we consider the overlapping criteria for the discovery of merging candidates to be the shared percentage of the larger bicluster. When multiple biclusters have overlapping areas, two criteria can be consider: to merge all rows and columns if all pairs of biclusters satisfy the considered overlapping criterion (relaxed setting) and to merge all rows and columns by comparing the shared area among all with the area of the larger bicluster (restrictive setting). We propose two procedures to efficiently deal with the merging of biclusters, one for each setting.

Maximal Circuits. The first proposed procedure, MergeCycle, is the combination of a graph search method with several heuristics to guide the search space exploration. Consider μ to be the overlapping degree between two biclusters. Since the overlapping degree is typically defined as the number of shared elements by the larger bicluster, heuristics can be defined assuming that bicluster are order by size. Consider two biclusters: a larger bicluster, (I, J), and a smaller bicluster (I', J'). If they do not satisfy $|I'|\times|J'| \leq \mu|I|\times|J|$, we do not need to compute their similarity, neither to compute the similarity for smaller biclusters than (I', J'). This is the first heuristic for pruning the search space. Second heuristic is to further prune the space by computing similarities along one dimension only. Pairs of biclusters not satisfying either $|I \cap I'| \geq \mu|I|$ or $|J \cap J'| \geq \mu|J|$ can be removed without computing the similarities for the remaining dimension. The chosen dimension is the one with average lower size among biclusters.

After computing the pairs of biclusters satisfying the overlapping threshold, a new procedure needs to be applied to verify the availability of larger candidates.

Illustrating, if (B_1, B_2), (B_1, B_3) and (B_2, B_3) are candidates for merging, then (B_1, B_2, B_3) is also a candidate. For this step, we map the candidate pairs of biclusters as an unweighted undirected graph, where the nodes are the biclusters and their links are given by the pairs. Under this formulation, the larger candidates for merging are edge-disjoint cyclic subgraphs (or circuits). This procedure is illustrated in Fig.3, where the cycles in the graph composed of candidate pairs are used to derive the three larger biclusters identified in Fig.2.

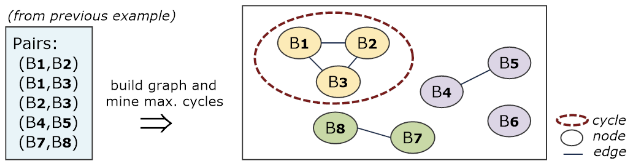

Fig.3: MergeCycle: computing merging candidates from pair candidates as a search for edge-disjoint cycles in graphs of biclusters.

Multi-support FIM. The second proposed merging procedure, MergeFIM, maps the task of merging biclusters as an adapted frequent itemset mining task. Let the elements of the original matrix be the available transactions, and the biclusters be the available items. Recovering the illustrative case presented in Fig.2, the (x_2, y_2) transaction would now have three items assigned, $\{B_1, B_2, B_3\}$. In this context, the support represents the number of shared elements for a specific set of biclusters (itemset). Understandably, for this scenario, we cannot rely on a general minimum support threshold, as the minimum number of shared elements to find a candidate for merging depends on the size of the larger bicluster. For this reason, the items (biclusters) are ordered by descending order of their size. When verifying if an itemset is frequent, instead of comparing its support with a minimum support threshold, the support is compared with the minimum support of the larger bicluster ($\mu|I|\times|J|$) that corresponds to the first item (1-length prefix) of the itemset. In this way, no computational complexity is added, and we guarantee that the output itemsets correspond to sets of biclusters that are candidates for merging.

This procedure follows three major steps. The first step is to create a minimal itemset database. Empty transactions are removed and transactions with one item can be pruned for further efficiency gains. Similarly to MergeCycle procedure, MergeFIM can also rely on the proposed heuristics to reduce the search space in order to produce the pairwise similarities. In this case, transactions with two items can be removed, and an Apriori-based method can be applied with the already 2-length itemsets derived from valid pairs of biclusters.

The second step is to run the adapted frequent itemset mining task using closed itemset representations. As previously described, such adaptation allows to replace the general notion of minimum by an indexable support based on the size of larger bicluster in the context of an itemset. Note that by using closed representation we avoid subsets of items with the same number of transactions. This means that the output of the mining task is precisely the set of merging procedures that are required.

The final step is to compose the new biclusters and, optionally, to derive the respective sequential patterns when required. This procedure is illustrated in Fig.4, and it follows the illustrative case introduced in Fig.2. The efficiency of this procedure is based on the observation that the mapped itemset databases tend to be highly sparse.

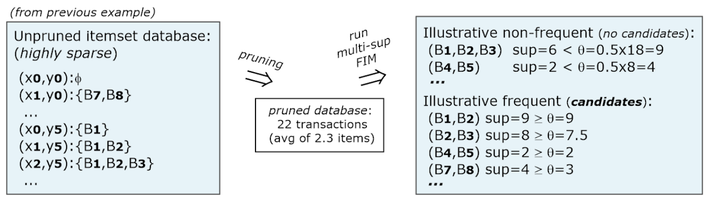
(from previous example)
Unpruned itemset database: (highly sparse)
(x0,y0):ϕ
(x1,y0):{B7,B8}
...
(x0,y5):{B1}
(x1,y5):{B1,B2}
(x2,y5):{B1,B2,B3}
...
pruning
pruned database: 22 transactions (avg of 2.3 items)
run multi-sup FIM
Illustrative non-frequent (no candidates):
(B1,B2,B3) sup=6 < θ=0.5x18=9
(B4,B5) sup=2 < θ=0.5x8=4
...
Illustrative frequent (candidates):
(B1,B2) sup=9 ≥ θ=9
(B2,B3) sup=8 ≥ θ=7.5
(B4,B5) sup=2 ≥ θ=2
(B7,B8) sup=4 ≥ θ=3
...

Fig.4: MergeFIM: computing merging candidates as a frequent itemset mining task.

4 Results and Discussion

This section evaluates the performance of IndexSpan and MergeIndexBic against competitive alternatives on synthetic and real datasets. IndexSpan and MergeIndexBic were implemented in Java (JVM version 1.6.0-24)[2]. We adopted PrefixSpan[3], still considered a state-of-the-art SPM method, and the OPC-Tree method [13] as the bases of comparison. The experiments were computed using an Intel Core i5 2.30GHz with 6GB of RAM.

4.1 Synthetic datasets

The generated experimental settings are described in Table 1. First, we created dense datasets (each item occurs in every sequence) by generating matrices up to 2.000 rows and 100 columns. Each sequence is derived from the ordering of column indexes for a specific row according to the generated values, as illustrated in Fig.1 from Section 2. Understandably, each of the resulting item-indexable sequences contains all the items (or column indexes), which leads to a highly dense dataset. Sequential patterns were planted in these matrices by maintaining the order of values across a subset of columns for a subset of rows. The number and shape of the planted sequential patterns were also varied. For each setting we instantiated 6 matrices: 3 matrices with a background of random values, and 3 matrices with values generated according to a Gaussian distribution. The observed results are an average across these matrices. The number of supporting sequences and items for each sequential pattern followed a Uniform distribution.

Matrix size (♯rows × ♯columns)	100×30	500×50	1000×75	2000×100
Nr. of hidden seq. patterns	5	10	20	30
Nr. rows for the hidden seq. patterns	[10,14]	[12,20]	[20,40]	[40,70]
Nr. columns for the hidden seq. patterns	[5,7]	[6,8]	[7,9]	[8,10]
Assumptions on the inputted thresholds	θ=5% δ=3	θ=5% δ=4	θ=5% δ=5	θ=5% δ=6

Table 1: Properties of the generated dataset settings.

[2] Software and datasets available in: http://web.ist.utl.pt/rmch/software/indexspan/
[3] Implementation from SPMF: http://www.philippe-fournier-viger.com/spmf/

Fig.5 compares the performance of the alternative approaches for the generated datasets in terms of time and maximum memory usage. Both PrefixSpan and OPC-Tree can be seen as competitive baselines to assess efficiency. Note that we evaluate the impact of mining sequential patterns in the absence and presence of the δ input (minimum number of items per pattern) for a fair comparison.

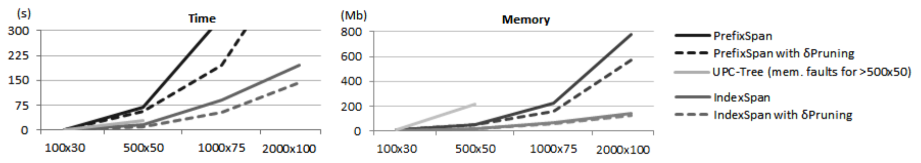

Fig.5: Performance of alternative SPM methods for datasets with varying properties.

Two main observations can be derived from this analysis. First, the gains in efficiency from adopting fast database projections are significant. In particular, the adoption of fast projections for hard settings dictates the scalability of the SPM task. Pruning methods should also be considered in the presence of the pattern length threshold δ. Contrasting with OPC-Tree and PrefixSpan, IndexSpan guarantees acceptable levels of efficiency for matrices up to 2000 rows and 100 columns for a medium-to-large occupation of sequential patterns ($\sim$3%-10% of matrix total area). Second, IndexSpan performs searches with minimal memory waste. The memory is only impacted by the lists of sequence identifiers maintained by prefixes during the depth-first search. Memory of PrefixSpan is slightly hampered due to the need to maintain the projected postfixes. OPC-Tree requires the full construction of the pattern-tree before the traversal, which turns this approach only applicable for small-to-medium databases. For an allocated memory space of 2GB, we were not able to construct OPC-Trees for input matrices with more than 40 columns.

To further assess the performance of IndexSpan, we fixed the 1000×75 experimental setting and varied the level of sparsity by removing specific positions from the input matrix, while preserving the planted sequential patterns. We randomly selected these positions to cause a heighten variance of length among the generated sequences. The amount of removals varied between 0 and 40%. This analysis is illustrated in Fig.6.

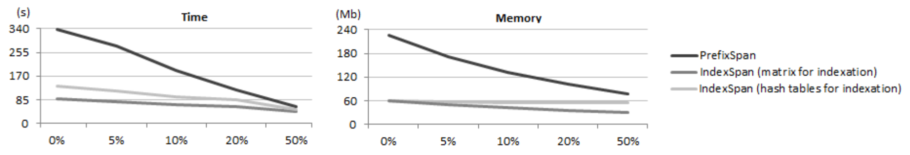

Fig.6: Performance for varying levels of sparsity (1000×75 dataset).

Two main observations can be retrieved. First, to guarantee an optimal memory usage, there is the need to adopt vectors of hash tables in IndexSpan. Second, although the use of these new data structures hampers the efficiency of IndexSpan, the observable computational time is still significantly preferable over the PrefixSpan alternative.

In order to assess the impact of varying the number of co-occurrences vs. precedences, we adopted multiple discretizations for the 1000×75 dataset. By decreasing the size of the discretization alphabet, we are increasing the amount of co-occurrences and, consequently, decreasing the number of itemsets per sequence. This analysis is illustrated in Fig.7. When the number of precedences per sequence is very small (<10), the efficiency tends to significantly decrease due to the exponential increase of sequential patterns. However, for the remaining discretizations, the efficiency does not strongly differ since the number of frequent patterns is identical and pattern-growth methods are able to deal with co-occurrences and precedences in similar ways (Algorithm 1 *lines 20-27*).

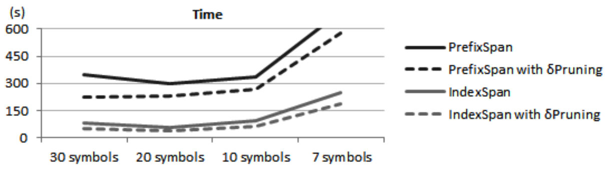

Fig.7: Performance for varying weights of precedences vs. co-occurrences.

To evaluate the relevance of considering noise-relaxations in order to compose larger sequential patterns, we selected the 1000×75 setting and exchanged the order of 5% of the items. Fig.8 traces item-indexable SPM performance for varying noise-relaxations by using MergeIndexBic with different overlapping degrees. Performance is measured using match scores based on the Jaccard index to assess: *1)* to what the extent do the found patterns match with planted patterns (correctness), and *2)* how well are the planted biclusters recovered (completeness). When relaxing the overlapping criteria, match scores increase, as the merging step allows for the recovery of order violations. However, this improvement in behavior is only observable until a certain overlapping threshold. The correct identification of this threshold can lead to significant gains (near 15 percentage points for this experiment).

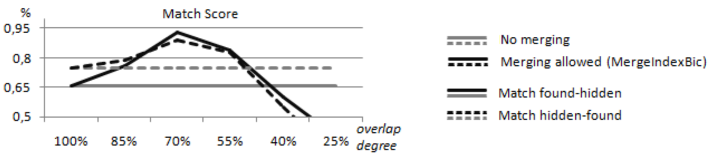

Fig.8: Impacting of merging sequential patterns in noisy contexts.

Finally, in order to show why MergeIndexBic is needed for an efficient merging step, we maintained the previous experimental settings and compared the performance of traditional combinatorial procedures (where similarities are computed for all pairs of biclusters, and the composition of larger candidates is recursively accomplished) against the proposed MergeCycle and MergeFIM procedures. MergeFIM procedure relies on the efficient Charm[4] method for the delivery of closed frequent itemsets. Fig.9 illustrates this analysis for varying overlapping degrees. Clearly, traditional procedures do not scale. MergeFIM outperforms MergeCycle for hard scenarios where there are large candidates for merging, a case that is commonly observed for relaxed levels of overlapping.

[4] Implementation from SPMF: http://www.philippe-fournier-viger.com/spmf/

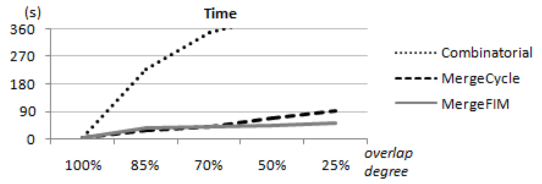

Fig.9: Comparing the efficiency of MergeCycle and MergeFIM against peer procedures.

4.2 Real datasets

To assess the performance of the proposed approaches in real datasets, we used multiple gene expression matrices[5]: dlblc (180 items per instance, 660 instances), coloncancer (62 items per instance, 2000 instances), leukemia (38 items per instance, 7129 instances). The goal is to discover order-preserved biclusters. For this purpose, we followed the procedure described in Fig.1 to generate the sequence databases using a discretization alphabet with 20 symbols. The positions corresponding to missing values were removed. Fig.10 compares the performance of the alternative approaches for the θ=8% and δ=5 thresholds. This analysis reinforces the previous observations. OPC-Tree is bounded by the size of the database. The adoption of IndexSpan strategies to deal with item-indexable sequences strongly impacts the SPM performance, and, consequently, the ability to discover order-preserving biclusters in real data.

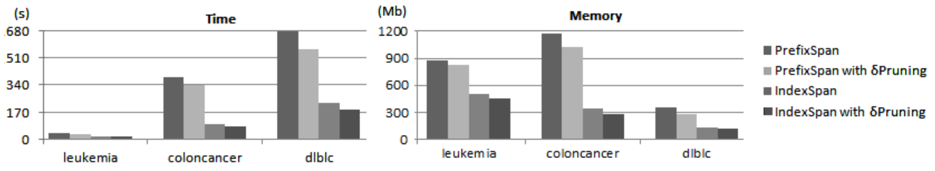

Fig.10: Performance of SPM-based order-preserving biclustering for biological data.

The relevance of tolerating noise to compose larger sequences is illustrated in Fig.11 for the leukemia dataset. Here, we computed the functional enrichment of the genes supporting each frequent sequence, Φ_s, recurring to the GoToolBox [16]. As a measure of significance, we counted the number of overrepresented terms with Bonferroni corrected p-values below 0.01. We observe that MergeIndexBic procedure increases not only the number of significant terms, but also their relative percentage as it removes short patterns from the output.

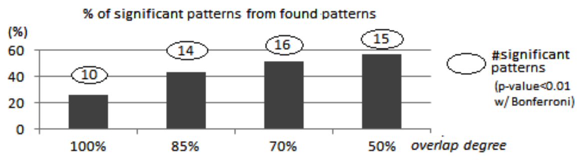

Fig.11: Biological relevance of allowing noise using MergeIndexBic for leukemia data.

[5] http://www.upo.es/eps/bigs/datasets.html
http://www.bioinf.jku.at/software/fabia/gene_expression.html

5 Conclusions

This work formalizes the task of performing noise-tolerant sequential pattern mining over item-indexable databases and motivates its relevance for a critical set of applications. The performance of existing approaches based on general SPM methods and on dedicated algorithms, such as the OPC-Tree, is discussed. To tackle the inefficiencies of existing solutions, we propose the IndexSpan algorithm. IndexSpan relies on position induction to deliver fast and memory-free database projections. Additionally, early-pruning techniques with impact on the performance of IndexSpan can be adopted to guarantee that only large sequential patterns are discovered.

Furthermore, we explore alternatives to efficiently extend IndexSpan in order to compose large sequential patterns under parameterizable noise allowance guarantees. MergeIndexBic is proposed to surpass the limited robustness and efficiency of existing options. This is done by merging sequential patterns with significant overlap on sequence items and on the supporting transactions. Pruning heuristics to avoid the computation of similarities among all pair are proposed. Efficient computation of candidates is achieved by mapping the merging task as maximal cycle discovery in undirected graphs or as multi-support frequent itemset mining.

Results on both synthetic and real datasets show the superior performance and relevance of the IndexSpan and MergeIndexBic methods.

Since the proposed item-indexable SPM relies on a prefix-growth search, it can easily accommodate principles from existing research to deliver condensed pattern representations, such as CloSpan [24], in order to reduce the complexity of the merging step; and to discover sequential patterns in distributed settings, such as MapReduce [23], in order to relax the efficiency boundaries of IndexSpan.

Acknowledgments

This is an extension of previous work [10] supported by *Fundação para a Ciência e Tecnologia* under the project D2PM (PTDC/EIA-EIA/110074/2009), project Neuroclinomics (PTDC/EIA-EIA/ 111239/2009), and PhD grant SFRH/BD/75924/2011.

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE. pp. 3–14. IEEE CS, Washington, DC, USA (1995)
2. Antunes, C., Oliveira, A.L.: Mining patterns using relaxations of user defined constraints. In: Knowledge Discovery in Inductive Databases (2004)
3. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: KDD. pp. 429–435. ACM, New York, USA (2002)
4. Bayardo, R.J.: Efficiently mining long patterns from databases. SIGMOD Rec. 27(2), 85–93 (1998)
5. Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z.: Discovering local structure in gene expression data: the order-preserving submatrix problem. In: RECOMB. pp. 49–57. ACM, New York, NY, USA (2002)
6. Cheng, H., Yu, P.S., Han, J.: Approximate frequent itemset mining in the presence of random noise. In: Maimon, O., Rokach, L. (eds.) Soft Computing for Knowl. Disc. and Data Mining, pp. 363–389. Springer (2008)

7. Chiu, D.Y., Wu, Y.H., Chen, A.L.P.: An efficient algorithm for mining frequent sequences by a new strategy without support counting. In: ICDE. pp. 375–. IEEE CS, Washington, DC, USA (2004)
8. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15(1), 55–86 (Aug 2007)
9. Han, J., Yang, Q., Kim, E.: Plan mining by divide-and-conquer. In: ACM SIGMOD IW on Research Issues in DMKD (1999)
10. Henriques, R., Madeira, S., Antunes, C.: Indexspan: Efficient discovery of item-indexable sequential patterns. In: ECML/PKDD IW on New Frontiers in Mining Complex Patterns (2013)
11. Kumar, P., Krishna, P., Raju, S.: Pattern Discovery Using Sequence Data Mining: Applications and Studies. Igi Global (2011)
12. Lin, D.I., Kedem, Z.M.: Pincer search: A new algorithm for discovering the maximum frequent set. In: EDBT. LNCS, vol. 1377, pp. 105–119. Springer (1998)
13. Liu, J., Wang, W.: Op-cluster: Clustering by tendency in high dimensional space. In: ICDM. pp. 187–. IEEE CS, Washington, DC, USA (2003)
14. Liu, J., Yang, J., Wang, W.: Biclustering in gene expression data by tendency. In: Comput. Systems Bioinformatics Conf. pp. 182–193. IEEE (2004)
15. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surveys 43(1), 3:1–3:41 (2010)
16. Martin, D., Brun, Remy, Mouren, Thieffry, Jacq, B.: Gotoolbox: functional analysis of gene datasets based on gene ontology. Genome Biology (12), 101 (2004)
17. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Trans. on Knowl. and Data Eng. 16(11), 1424–1440 (2004)
18. Pei, J., Han, J., Mortazavi-Asl, B., Zhu, H.: Mining access patterns efficiently from web logs. In: PADKK. pp. 396–407. Springer-Verlag, London, UK, UK (2000)
19. Raedt, L.D., Guns, T., Nijssen, S.: Constraint programming for data mining and machine learning. In: AAAI. AAAI Press (2010)
20. Salvemini, E., Fumarola, F., Malerba, D., Han, J.: Fast sequence mining based on sparse id-lists. In: ISMIS. pp. 316–325. Springer-Verlag, Berlin, Heidelberg (2011)
21. Serin, A., Vingron, M.: Debi: Discovering differentially expressed biclusters using a frequent itemset approach. Algorithms for Molecular Biology 6, 1–12 (2011)
22. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT. pp. 3–17. Springer-Verlag, London, UK (1996)
23. qing Wei, Y., Liu, D., shan Duan, L.: Distributed prefixspan algorithm based on mapreduce. In: Inf. Tech. in Medicine and Education. vol. 2, pp. 901–904 (2012)
24. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Datasets. In: SDM. pp. 166–177 (2003)
25. Yang, J., Wang, W., Yu, P.S., Han, J.: Mining long sequential patterns in a noisy environment. In: SIGMOD. pp. 406–417. ACM, New York, NY, USA (2002)
26. Yang, Z., Wang, Y., Kitsuregawa, M.: Lapin: effective sequential pattern mining algorithms by last position induction for dense databases. In: DASFAA. pp. 1020–1023. Springer-Verlag, Berlin, Heidelberg (2007)
27. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Mach. Learn. 42(1-2), 31–60 (2001)
28. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: WWW. pp. 791–800. ACM (2009)
29. Zhu, F., Yan, X., Han, J., Yu, P.S.: Mining Frequent Approximate Sequential Patterns. Chapman&Hall (2009)
30. Zhu, F., Yan, X., Han, J., Yu, P., Cheng, H.: Mining colossal frequent patterns by core pattern fusion. In: ICDE. pp. 706–715 (2007)