IndexSpan: Efficient Discovery of Item-Indexable Sequential Patterns

Rui Henriques, Sara C. Madeira, and Cláudia Antunes

Dep. of Comp. Science and Eng. *and* KdBIO, Inesc-ID
Instituto Superior Técnico, University of Lisbon, Portugal
{rmch,sara.madeira,claudia.antunes}@ist.utl.pt

Abstract. The research on sequential pattern mining has been driven by efficiency principles. However, efficiency is still a critical drawback for tasks that require the discovery of sequential patterns with medium-to-large length. Many of these tasks, such as pattern-based biclustering, rely on datasets with item-indexable properties. An item-indexable database, typically observed in order-preserving datasets across biological and customer-service domains, does not allow item repetitions per sequence. In this work, we propose a new sequential pattern mining method, called IndexSpan, which is able to mine sequential patterns over item-indexable databases with heightened efficiency in comparison with the existing alternatives. The superior performance of IndexSpan is demonstrated on both synthetic and real datasets, and its relevance for multiple applications is discussed.

1 Introduction

During the last two decades, a few number of methods have been proposed to deal efficiently with the discovery of sequential patterns. These methods are prepared to deal with sequence databases with an arbitrary repetition of items per sequence. However, many real-world tasks rely on a more restricted form of sequences, item-indexable sequences, which assume simple ordering constraints among a subset of the overall items. To guarantee the consistency of ordering constraints among items, item-indexable sequences do not allow item repetitions. Illustrative examples of such databases include sequences derived from consumer ratings, shopping items ordered by the time of acquisition, schedule of tasks, order-preserving gene expression values in microarrays, among many others. Although some of these databases have varying levels of sparsity, item-indexable sequences derived from ratings and microarrays tend to be highly dense.

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. Although existing sequential pattern mining (SPM) approaches can be applied over these databases, they still show inefficiencies to deliver large patterns due to the combinatorial explosion of sequential patterns under low support thresholds [8]. Although there are principles for the discovery of colossal patterns based on the fusion of smaller itemsets [19], these approximations suffer from noise and, to our knowledge, have not been extended for the SPM task. Additionally, the few dedicated methods able to discover sequential patterns in item-indexable databases [7, 8] show significant memory overhead.

This work proposes a new method for the efficient discovery of sequential patterns over item-indexable sequences. This is done by using efficient data structures that keep track of the position of items per sequence and by relying on fast database projections based on the relative order of items. Additional optimizations are proposed based on pruning techniques when the user has only interest in sequential patterns above a minimum length.

The paper is structured as follows. *Section 2* introduces and motivates the SPM task over item-indexable databases, and covers existing contributions with potential relevance for its solution. IndexSpan, the proposed SPM method prone to deal with item-indexable databases, is described in *section 3*. In *section 4*, the performance of IndexSpan is assessed on both real and synthetic datasets against peer algorithms. 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 $\exists_{1 \leq i_1 < .. < i_n \leq m}$: $a_1 \subseteq b_{i_1}$,..,$a_n \subseteq b_{i_n}$. A *sequence database* is a set of sequences $D = \{s_1, .., s_n\}$.

The illustrative sequence s_1=<$\{a\}, \{be\}$>=$a(be)$ is contained in s_2=$(ad)c(bce)$ and is maximal w.r.t. to D=$\{ae, (ab)e\}$.

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 $|\Phi_s|$.

To illustrate the previous concepts, consider the following 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 let the length of a sequence $|s|$ be the number of item occurrences, $\Sigma_i\ |s^i|$. Given a set of item-indexable sequences D, a minimum support threshold θ, and a minimum sequence length δ. The target task of **SPM over item-indexable sequences** 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

Some of the most prominent applications for SPM task that rely on item-indexable databases include:

- order-preserving biclustering, a form of local clustering, that relies on sequential patterns [7, 3]. This type of biclustering is commonly applied over biological data, including gene expression and genomic structural variation analysis. To compose the database from real-value or discrete matrices, the column indexes are linearly ordered for each transaction according to their values. Each transaction is, consequently, seen as a sequence of items that correspond to column indexes. Biclusters are derived from the set of items and supporting transactions for each sequential pattern, as illustrated in Fig.1;
- pattern-centric analysis of recommendations based on user preferences and of service quality based on questionnaires [6]. In these scenarios, item-indexable sequences are derived from an ordering of user ratings. Interestingly, frequent precedences and co-occurrence disclose important priorities or similarities across the evaluated aspects for different sets of users;
- other key applications, such as frequent scheduling/planning, shopping, and traveling behavior [18, 5].

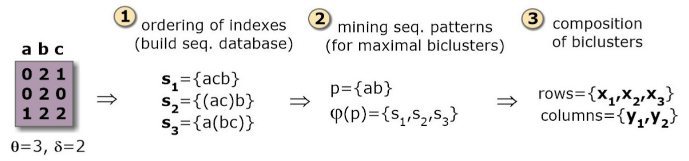
1 ordering of indexes (build seq. database)
2 mining seq. patterns (for maximal biclusters)
3 composition of biclusters
a b c
0 2 1
0 2 0
1 2 2
θ=3, δ=2
⇒
s1={acb}
s2={(ac)b}
s3={a(bc)}
⇒
p={ab}
φ(p)={s1,s2,s3}
⇒
rows={x1,x2,x3}
columns={y1,y2}

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

2.2 Related Work

Although general SPM methods are not optimized to deal with item-indexable specificities, they have been the largely adopted to solve these applications [8]. 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 [9]. Apriori-based algorithms [13], and vertical-based variations [17], 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, alternatives such as the use of bitmaps or direct comparison have been proposed [4, 2].

Pattern growth methods [10, 11, 2] avoid the Apriori-based costs from candidate generation by building an expressive representation of the database and by recursively traversing it to grow the frequent sequences. PrefixSpan [10], one of the most efficient options to SPM, recursively constructs patterns by growing their prefix and by maintaining their corresponding postfix subsequences into projected databases. In the absence of gap-based constraints, 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 the construction of projected databases.

Early-pruning methods [16, 4, 12] emerged more recently in the literature. 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, thus avoiding support counting and generation of infrequent candidates.

The drawback of these SPM alternatives is that their performance does not scale for very low support thresholds, which is 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 [7, 8] 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.

3 Solution: IndexSpan

To avoid the drawbacks of existing approaches, we propose the IndexSpan algorithm, an extension of PrefixSpan to discover sequential patterns with heightened efficiency from item-indexable sequence databases.

Comparison of existing SPM algorithms [9] 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 [10] 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.

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 can be replaced by a vector of hash tables to optimize memory usage. Considering an illustrative sparse database with $|\mathcal{L}|$=20,

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 supporting 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 $D_\beta$;
```

D=$\{a(cp), am, (ac)\}$. Instead of relying on a 20×3 matrix to track the indexes [[0 -1 1 ...][...][...]], it is enough to monitor the positions of the available items [h_1{a:0,c:1,p:1},h_2{a:0,m:1},h_3{a:0,c:0}] to obtain the index (e.g. h_2(m)=1).

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.

4 Results

In this section, we evaluate the performance of IndexSpan and competitive alternatives on synthetic and real datasets. IndexSpan was implemented in Java (JVM version 1.6.0-24). We adopted PrefixSpan[1], still considered a state-of-the-art SPM method, and the OPC-Tree method [7] 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 items 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. The number of supporting sequences and items for each sequential pattern followed a Uniform distribution over the ranges presented in Table 1.

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 results are an average across these matrices. The properties of the generated databases and planted patterns are detailed in Table 1.

Fig.2 compares the performance of the alternative approaches for these settings 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, the minimum number of items per pattern, for a fair comparison.

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

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

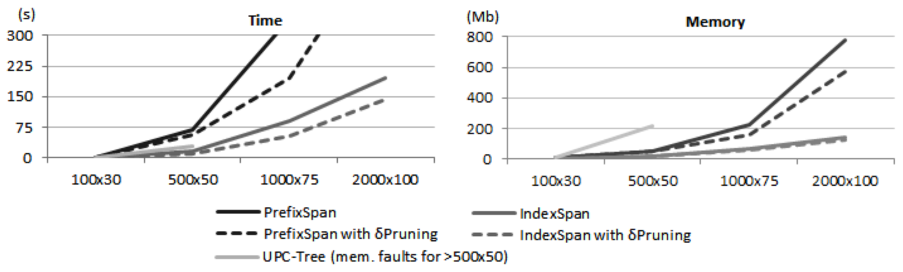

Fig.2: Performance of alternative SPM methods for datasets with varying properties.

Two main observations can be derived from this analysis. First, the gains in efficiency from adopting fast database projections are very 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 (∼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 on 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 was varied between 0 and 40%. This analysis is illustrated in Fig.3.

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.

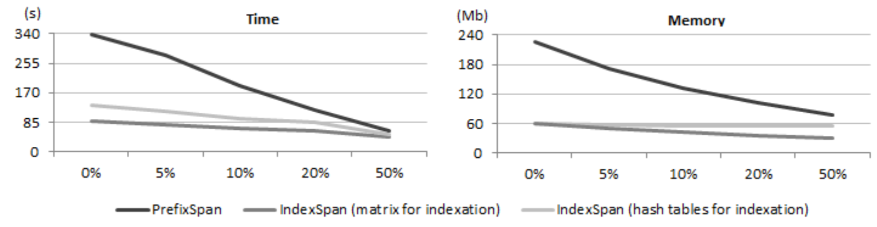

Fig.3: Performance for varying levels of sparsity (1000×75 dataset).

Finally, 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.4. 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*).

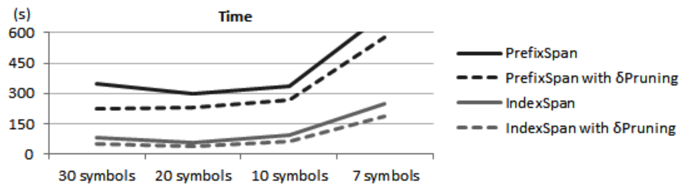

Fig.4: Performance for varying weights of precedences vs. co-occurrences.

4.2 Real datasets

To assess the performance of the target approaches in real datasets, we adopted multiple gene expression matrices[2]: 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.5 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 se-

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

quences strongly impacts the SPM performance, and, consequently, the ability to discover order-preserving biclusters in real data.

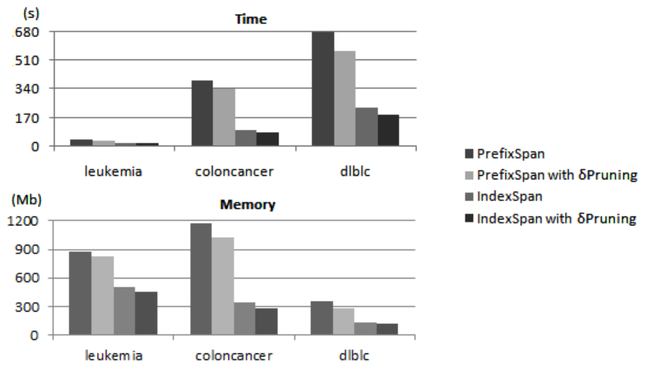

Fig.5: Performance of SPM-based order-preserving biclustering for biological data.

5 Discussion

This work formalizes the task of performing 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 UPC-Tree, are 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.

Since IndexSpan relies on a pattern-growth search, it can easily accommodate principles from existing research to deliver alternative pattern representations or to discover sequential patterns in distributed settings. Potential directions include the incorporation of principles of methods that extend PrefixSpan to discover condensed patterns, such as CloSpan [15], or to parallelize the mining task in distributed settings, such as MapReduce [14].

Results on both synthetic and real datasets show the superior performance of the proposed method. IndexSpan makes an optimal use of memory and is able to achieve significant improvements in computational time on both sparse and dense datasets.

Acknowledgments

This work was supported by *Fundação para a Ciência e Tecnologia* under the research project D2PM, PTDC/EIA-EIA/110074/2009, and the 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. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: ACM SIGKDD. pp. 429–435. ACM, New York, NY, USA (2002)
3. 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)
4. 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)
5. Han, J., Yang, Q., Kim, E.: Plan mining by divide-and-conquer. In: ACM SIGMOD IW on Research Issues in DMKD (1999)
6. Kumar, P., Krishna, P., Raju, S.: Pattern Discovery Using Sequence Data Mining: Applications and Studies. Igi Global (2011)
7. Liu, J., Wang, W.: Op-cluster: Clustering by tendency in high dimensional space. In: ICDM. pp. 187–. IEEE CS, Washington, DC, USA (2003)
8. Liu, J., Yang, J., Wang, W.: Biclustering in gene expression data by tendency. In: Comput. Systems Bioinformatics Conf. pp. 182–193. IEEE (2004)
9. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surveys 43(1), 3:1–3:41 (2010)
10. 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)
11. 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)
12. 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)
13. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT. pp. 3–17. Springer-Verlag, London, UK, UK (1996)
14. 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)
15. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Datasets. In: SDM. pp. 166–177 (2003)
16. 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)
17. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Mach. Learn. 42(1-2), 31–60 (2001)
18. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: IC on WWW. pp. 791–800. ACM, New York, NY, USA (2009)
19. Zhu, F., Yan, X., Han, J., Yu, P., Cheng, H.: Mining colossal frequent patterns by core pattern fusion. In: ICDE. pp. 706–715 (2007)