Università di Pisa
Dipartimento di Informatica

Technical Report: TR-09-08

Sequential Pattern Mining with Temporal and Content Constraints

Miriam Baglioni Elena Ratti Franco Turini

April 8, 2009

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Sequential Pattern Mining with Temporal and Content Constraints

Miriam Baglioni Elena Ratti Franco Turini

April 8, 2009

Abstract

A tool for extracting sequential patterns with temporal and content constraints from logs is presented. The heart of the tool is a new algorithm capable of handling temporal constraints and itemsets with multiple occurrences of the same item. Applications of the system are discussed in the context of monitoring logs registered by a platform supporting distributed processes, developed within the European project BRITE.

1 Introduction

In many application contexts, processes, which need a distributed interactive execution, are supported by an infrastructure capable of handling the transmission of data and requests. The logging of actions performed by means of the infrastructure can provide a collection of information extremely useful for monitoring the application processes. More than that, suitable analysis techniques can provide hints for discovering anomalous behaviors and opportunities for redesigning the processes themselves. The principal characteristic of the logs is that they contain both content related information and time related information. As a consequence the analysis tools need to handle both features related to the semantics of logged actions and features related to the time flow.
The goal of this paper is to present a tool supporting knowledge discovery from logs. The heart of the tool is an algorithm for extracting frequent sequential patterns keeping temporal constraints into account and allowing itemsets with multiple occurrences of the same item and with timestamps bound to each item. Our work was inspired by problems coming out in the European IP project BRITE[1]. The general goal of the project is to set up an innovative infrastructure for supporting the European Business Registers, that is the offices in charge of registering the operations of private companies, e.g. changing the seat, maybe from one country to another one, or opening a branch, again possibly in another country, and so on. The BRITE platform is also required to offer a dashboard for monitoring the processes and offering the possibility of performing analyses on the logged actions. One of the purposes of the analysis is to

discover anomalous behaviors, for example for supporting the anti money laundering actions required by the European Union. As an example, the opening of multiple companies in a row by the same legal entity.
Another purpose is the detection of possible bottlenecks in the processes and the discovery of the possibilities of streamlining procedures.

The paper is organized as follows: Sec.2 reports on the state of the art about algorithms for mining frequent sequential patterns. Sec. 3 presents our algorithm while Sec. 4 discusses its performance in terms of execution time and extracted patterns. Sec. 5 discusses the handling of content constraints and Sec.6 reports some conclusions and ideas for future extensions.

2 Related Work

Many studies have contributed to the efficient mining of sequential patterns or other frequent patterns in time-related data. There are essentially two methods for mining sequential patterns: Apriori based and pattern growth methods. These two families differ mostly for the way the patterns are discovered. Apriori-like algorithms [4, 8, 6, 12, 9, 13] are based on the Apriori principle [3], with all its strengths and weaknesses. Among the weaknesses there is the huge amount of candidate patterns that can be generated on large sequence databases, and the need of multiple database scans during the mining. These two problems can be tackled by pattern growth methods. The first proposed pattern growth method was FreeSpan [10], which reduces the efforts of candidate subsequence generation. Han et al. proposed a second and more efficient method, called PrefixSpan [11].It is based on the growth of the prefix (the sequential pattern found at the previous step) by appending an item x to it, if this item occurs at least a number of times greater than minimum support. Notice that the item has to be searched only over sequences where the prefix occurs. In general a i-length frequent sequential pattern p can be grown by any item found frequent over the set of the sub-sequences that are p-suffix. This set of sequences is called the projected database of prefix p. An item x can be appended to the prefix in two different ways: serial and parallel. Serial means that the prefix is grown appending a new itemset containing that item; parallel means that the prefix is grown by appending the item x to the last itemset of the prefix. In order to deal with gap constraints a revised version of PrefixSpan has been proposed in [5] as the GenPrefixSpan algorithm: a generalization of prefix-growth methods for sequential pattern mining with gap constraints. GenPrefixSpan is based on the redefinition of the method used to construct projected databases: instead of looking only for the first occurrence of the item used to grow the prefix, every further occurrence is considered. An evolution of this algorithm was proposed by the same authors as GenPrefixGrowth[7] which, besides gap constraints, deals with temporal gap and content constraints (generated by a context-free language). When the sequence database is composed of a significant number of sequences with the same element repeated several times, GenPrefix-

Growth projection method (the same as in GenPrefixSpan when dealing with gap constraints) becomes prohibitive in terms of memory. It happens because the projected database for each different element may be much bigger than the original database, thus violating the assumption that the size of the projected database cannot exceed that of the original one, as it is the case of the original version of PrefixSpan. A projection method that tackle this problem is that proposed in the PrefixTime algorithm[**?**]. PrefixTime deals with temporal gap constraints guaranteeing that the size of the projected database does not become greater than the one computed in the previous step. The projection idea of PrefixTime is based on the "alternative prefix search". Given a projected database related to a prefix p and a sequence s in it and a locally frequent item x, when PrefixTime tries to grow the prefix p with x it searches for the first occurrence of x in s. If it finds it and the current x occurrence does not satisfy the gap constraints, then PrefixTime searches for a further x occurrence until a valid x occurrence is found or the end of the sequence s is reached. In this last case PrefixTime tries to search an alternative prefix for each x occurrence found in the previous s scan. The search for an alternative prefix is done backward in s starting from the itemset which contains the item x considered in that moment. We chose to develop a prefix-growth based method because such methods have no candidate generation phase: they only grow longer sequential patterns from the shorter frequent ones. Furthermore, sequences shrink quickly: generally a projected database is smaller than the original one because only the suffix sub-sequences of a frequent prefix are projected into a projected database, and the number of sequences where current x item occurs is smaller or equal than the size of the projected database at the previous step. These are the main reasons why, having to handle huge sequence databases, we focused on prefix growth methods. In particular we propose an algorithm based on the PrefixTime "alternative prefix search" idea. Our algorithm differs from PrefixTime on the notion of itemset and on the use of a max window time constraint.

3 The Algorithm

SPMTCC (Sequential Pattern Mining with Temporal and Content Constraints) is a prefix-growth algorithm for the discovery of sequential patterns with existential, temporal and content constraints. On the discovered patterns some constraints can be specified: a sequence can become a valid pattern only if satisfies the constraints. In SPMTCC, existential constraints are expressed by specifying a minimum or maximum support; temporal constraints are expressed by specifying a value for the following gap: min-gap (minimum temporal gap between two consecutive itemsets), max-gap (maximum temporal gap between two consecutive itemsets), max-span (maximum temporal gap between the first and the last itemset of the patterns to be extracted), max-win (maximum temporal distance between items of the same itemset). Temporal gap constraints are exemplified in Figure 1 Temporal gaps can have different time granularity which can be: hour, day, week , month , year.

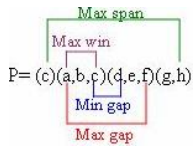

Figure 1: Temporal gap constraints.

Temporal gap constraints, on which the relations $mingap < maxgap$, $maxgap < maxspan$ and $maxwin < maxgap$ must hold, are dealt within the mining process. Content constraints are dealt in the post processing phase.

Before describing the algorithm, we briefly recall the basic general concepts. An item is an element from a totally ordered set Σ (the items collection). Items can be "aggregated" within an itemset that is defined as a non empty sequence of distinct items. A sequence is an ordered list of itemsets. When we deal with temporal data the concept of itemset is included in that of event, which is a pair composed of an itemset and a timestamp. When we talk about sequences as ordered lists of events also the timestamps associated to the events in the sequence must be ordered ($\alpha =< a_1, a_2, \ldots, a_n > \forall i \epsilon [1, n-1].a_{i.Time} < a_{i+1.Time}$). The concepts of subsequence and maximality can be defined among the sequences of itemsets. A sequence α is said to be a *subsequence* of another sequence β if for all itemsets in α there exist an itemset in β which contains the one in α maintaining the ordering. A sequence α is said to be *maximal* in a set S of sequences if it does not exist in S any sequence β such that α is a subsequence of β.
The following two definitions are a slightly different version of those in [11]. Given a sequence $\alpha =< e_1, e_2, \ldots, e_n >$ (where each e_i corresponds to a frequent itemset in database S), a sequence $\beta =< e'_1, \ldots, e'_m > (m \leq n)$ is called a *prefix* of α if and only if 1) $e'_i = e_i$ $(i \leq m-1)$; 2) $e'_m \subseteq e_m$; Given a sequence $\alpha =< e_1, e_2, \ldots, e_n >$ (where each e_i corresponds to a frequent itemset in database S), let $\beta =< e_1, e_2, \ldots, e_{m-1}, e'_m > (m \leq n)$ be the prefix of α. Sequence $\gamma =< e''_m, e_{m+1}, \ldots, e_n >$ is called the *suffix* of α with regards to prefix β, where $e''_m = (e_m - e'_m)$. Note that, if β is not a subsequence of α, the suffix of α with respect to β is empty.

The main differences with respect to other algorithms are the notion of itemset and the way of dealing with window-size constraints. We consider an itemset as a set of items where 1) each item has its timestamp, 2) items can have different timestamps, 3) two items with the same value and/or the same timestamp can coexist, 4) items are time ordered, 5) the time difference between the last and the first item within an itemset is $\leq$ max-win.
Example: let time-granularity = day and max-win = 3 days, ($a_{11Nov08}$, $a_{11Nov08}$, $b_{11Nov08}$, $c_{12Nov08}$, $a_{12Nov08}$, $b_{13Nov08}$, $c_{14Nov08}$) is an itemset.

SPMTCC takes a file in input where each row has the form: customer id,

```
projectDBminsupp(Database db)
{
    for each serial frequent item x ∈ db do
    {
       ρ' = ρ with x serially appended
       ρ'_db = new database with prefix ρ'
       for all s in db
       {
          if serial_projected s then
          {
             add projected s to ρ'_db
             if min_supp is not reached then exit loop
          }
       }
       if (size(ρ'_db) ≥ min_supp then projectDBminsupp(ρ'_db)
       deleted(ρ'_db)
    }//end for serial projection

    for each parallel frequent item x ∈ db do
    {
       ρ'' = ρ with x appended in parallel
       ρ''_db = new database with prefix ρ''
       for all s in db
       {
          if parallel_projected s then
          {
             add projected s to ρ''_db
             if min_supp is not reached then exit loop
          }
       }
       if (size(ρ''_db) ≥ min_supp then projectDBminsupp(ρ''_db)
       deleted(ρ''_db)
    }//end for parallel projection
}
```

Figure 2: Algorithm Pseudo code

item, timestamp (in terms of date and time). This file is pre-processed and sequences are built by grouping rows by customer id. Each sequence is composed by a list of 1-length itemsets.
The next step is the call to a recursive procedure that performs the projection of the input database. SPMTCC includes two kinds of projection procedures for searching frequent or infrequent sequential patterns.
The following procedure refers to the case of frequent sequential pattern search, where a value for min support is specified. At the beginning db is the database built by the pre-processing phase. At each recursive step the prefix is grown by appending to it, either in serial or in parallel, a frequent item. In order to apply the alternative prefix approach, each sequence in each projected database has the following mining variables associated: **lastseed** that is the pointer to the last prefix itemset, **lastseedelem** that is the pointer to the last item of lastseed itemset, **firstseed** that is the pointer to the first itemset prefix, and **firstseedelem** that is the pointer to the first item of firstseed itemset.
In the following we call a sequential pattern valid if it satisfies the support and all temporal constraints. Let ρ be a valid sequential pattern computed at the nth recursive step and ρ_db be its projected database. Let x be a frequent item in ρ_db. Two databases are created: $\rho^{'}_db$ and $\rho^{''}_db$ for containing all the projected sequences: in the first case related to the pattern $\rho^{'}$ given by prefix ρ with x appended in parallel, and in the second case related to the pattern $\rho^{"}$ given by prefix ρ with x serially appended, $\rho^{'}_db$ ($\rho^{''}_db$) is filled with ρ_db sequences where a valid pattern $\rho^{'}$ ($\rho^{''}$) occurs.

"Serial frequent items", denoted by the **freq S** set, are those items that verify the minimun support and that may be serially appended to the prefix and are searched from the itemset that follows *lastseed* until the end of the sequence.

"Parallel frequent items", denoted by the **freq P** set, are those items that can be appended in parallel to the prefix. **freqP** contains both the items in **freqS** and those ones the occurrence of which, forward wrt *lastseedelem* and backward wrt *lastseed* (in this last case the items have to have the same timestamp as *lastseedelem*), respect the minimum support.
The serial projection of a sequence S is performed by searching for a valid pattern composed of the prefix appended with item x. All the possible values for x and possibly alternative prefixes are checked until a valid pattern is found or no more combinations are possible. When a valid pattern is found, the mining variables associated to a copy of S are modified in order to take into account the information about the new prefix. The same holds for the parallel projection of S, except for but the items being appended in parallel.

Execution example Consider the input db of Figure 3 (the first on the left of the Figure) and the following gap constraints: max-win = 3, min-gap = 4, max-gap = 10, max-span = 12 and min-sup = 3.

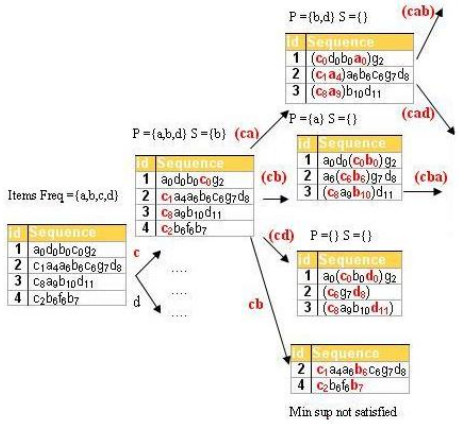

Figure 3: Example of execution.

First of all the set of frequent items is found over the input table and, for each of them, an operation of projection is performed. For the sake of the example and just to present the method, we show only some of the operation performed for growing the item (prefix) c. The second table from the left in Figure 3 shows the projected db computed for the prefix c. The sets **S** and **P** above the table represent the sets of frequent serial and parallel items. Note that a is a candidate to be appended in parallel to the prefix because it occurs forward in sequences 2 and 3, and backward in sequence 1 with the same timestamp of c. Thus, putting it after c_0 produces no loss of temporal information, so sequence 1 is updated exchanging the position of a_0 and c_0 to respect the order of the pattern *(ca)*, thus pattern ca is a valid pattern. The items between a_0 and c_0 are left in the same itemset of the new prefix *(ca)*. Consider now the projection

step for *(cb)*. The first occurrence in sequence 2 after c_1 of item b (b_6) cannot be appended in parallel because the max-win constraint is not satisfied, so we search for an alternative prefix for b_6. First we search for an alternative lastseed item (c) around b_6. Searching backward from b_6, we do not find any item c that satisfies the temporal constraints with b_6 while searching forward, we find the item c_6 that has the same timestamp of b_6. So sequence 2 is updated by exchanging b_6 and c_6, putting this two items together in the same itemset, and erasing all the items that do not have the same timestamp of lastseedelem. The only item that is not erased is a_6, which is used to validate the pattern (cba). b_6 becomes the new lastseedelem and c_6 the new firstseedelem, while the item c_1 is no more considered for this prefix in sequence 2. For the mining of pattern cb note that in sequence 1 there is no occurrence of b after prefix c_0 and in Sequence 3 the only occurrence of b is b_{10}, but it satisfies max-win with respect to prefix c_8, so it can be appended in parallel only. No alternative item c exists for b_{10}. Thus only in sequences 2 and 4 we found a valid pattern cb but they are not enough to satisfy the min sup constraint. All the other projection steps follow the same reasoning seen so far.

4 Testing the algorithm

SPMT(C)C was tested on data obtained from the IBM Generator. We tested

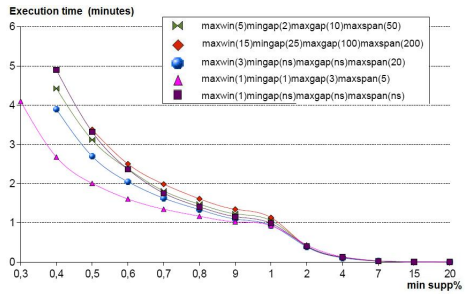

Figure 4: Execution time.

the algorithm on different data varying the number of sequences (from 1000 to 10000) and the number of distinct items (from 50 to 10000), while the average number of itemsets per sequence and the average itemset length were set as fixed values (10 and 2.5). Figures 4 and 6 represent the algorithm average behavior and show, respectively, the algorithm execution time and the number of extracted patterns. The input file was composed of 1000 sequences with 250 distinct items.

The first series of graphs shows the mining time w.r.t. minsup (%), varying the temporal gap constraints, while the second series of graphs illustrates the number of sequential patterns returned w.r.t. minsup(%), varying the temporal

gap constraints. We can note that the number of mined patterns depends on the size of maxwin: by increasing maxwin we can extract more patterns because of the higher number of combinations of the items within the itemsets. The

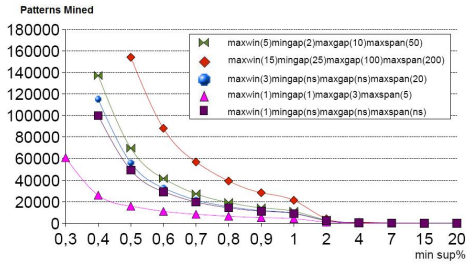

Figure 5: Number of extracted patterns.

execution time, instead, depends strongly on the distribution of the items within the sequences and on their timestamp (this influences the number of times the algorithm has to search for alternative prefixes), obviously weighted by the temporal constraints settings. You can, in fact, notice that the line with a square symbol, whose settings are considered always verified by the algorithm for all the gap but maxwin, and the line with a triangle symbol, whose settings for all the gap constraints are specified and the one for maxwin is equal to the one of the line with a square symbol, differ for more than two minutes when considering a support of about 0.4%. Note that we do not report any comparison w.r.t. PrefixTime because we include in our code a call to PrefixTime when the maxwin constraint is equal to 0. The PrefixTime code we link is a slightly modified version of the original one, which allows for multiple occurrences of the same item within an itemset.

5 Content constraint definition and inference rules

Content constraints are handled in a post-processing phase according to rules that specify the acceptable semantics of itemsets while respecting the sequentiality of the actions found by the algorithm. Given that an elementary constraint is a literal denoting the occurrence of an action, the content constraints can be defined as follows: Given two content constraints C_1 and C_2, then

- $C_1 \wedge C_2$ is a content constraint
- $C_1 \vee C_2$ is a content constraint
- $\neg C_1$ is a content constraint
- (C_1) is a content constraint

Given a pattern P and a constraint C, the validity of P with respect to C is given by a set of rules. These rules take a pattern P and a constraint C and return True (T = P verifies C) or False (F = P does not verify C). In the following, because of space constraints, we show only the definition for the $\wedge$ rule:

$$\frac{C{=}C_1{\wedge}C_2, <P,C_1>{\rightarrow}F}{<P,C>{\rightarrow}F}$$

If P does not verify C_1 and $C = C_1 \wedge C_2$ than P does not verify C.

$$\frac{\begin{array}{l} C = C_1 \wedge C_2, \\ < P, C_1 > \rightarrow T, \\ < P^{'}, C_2 > \rightarrow F, \\ P^{'} = P/\{p_i | \exists mint. <p_1 \ldots p_t, C_1> \rightarrow T, 0 < i \leq t\} \end{array}}{<P,C> \rightarrow F}$$

If P verifies C_1 but does not verify C_2 and $C = C_1 \wedge C_2$ than P does not verify C.

The last rule for the operator $\wedge$ directly follows. Notice that we need to consider $P^{'}$ because the constraints have to be verified in the order in which they are given. Consider for example the constraint $C = (a \wedge b) \wedge (d \vee f)$ and the pattern $P = dceaqbeg$. In this case $C1 = (a \wedge b)$ and $C2 = (d \vee f)$. So one pattern that verifies this constraint must contain the symbol a followed by b followed by one among d or f. P verifies C1 and $P^{'} = eg$ does not verify C2. If we had considered P when evaluating C2, the result would have been different since d is present in P.

The following is an example of the application of SPMTCC with its post-processing features on data (logs) recorded by the BRITE platform. The input

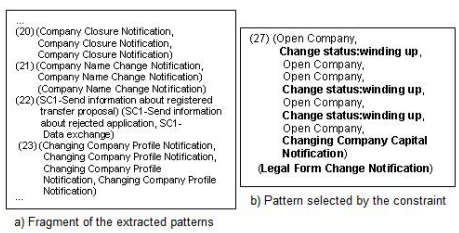

Figure 6: Patterns.

file is composed of 71 sequences obtained by aggregating 200 logs rows by the ID of the legal person associated to the company that performed the action. We executed the algorithm with several temporal gap settings, and we show the result of the one with maxwin = 28 days. SPMT(C)C mined 1939 patterns of which 34 were maximal. A fragment of the algorithm result is shown in Figure 6a). Over this patterns we apply the constraint $open \wedge open \wedge open$, which refers to a specific money laundering checking, and we obtained the pattern in Figure 6b). We can use the information given by the actions that are not open, to characterize a new possible malicious behavior, or to better specify the one we already know.

6 Conclusions and future work

The capability of using both very general time constraints and very general content constraints in mining sequential patterns can be very useful in many application contexts, when supported by a good performing system. We are currently addressing another application context stemming from another European Project, MUSING [2], where one of the goals is to provide tools for supporting the evaluation of operational risk for small-medium enterprises. One of the possible scenarios is controlling the failures of the network infrastructure of the company, and we are studying how to use our tool for performing suitable analyses of the logs recorded by the network infrastructure.
We are furthermore extending the system with a pre-filtering tool used for selecting, at loading time, sequences of logs complying with general content and temporal constraints.

References

[1] BRITE EU Project IST-27190, http://www.briteproject.net.

[2] MUSING EU Project FP6 027097, http://www.musing-project.eu.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[5] Cláudia Antunes and Arlindo L. Oliveira. Generalization of pattern-growth methods for sequential pattern mining with gap constraints. In Petra Perner and Azriel Rosenfeld, editors, *MLDM*, volume 2734 of *Lecture Notes in Computer Science*, pages 239–251. Springer, 2003.

[6] Claudia Antunes and Arlindo L. Oliveira. Sequential pattern mining algorithms: Trade-offs between speed and memory. Proceedings of the Second Workshop on Mining Graphs, Trees and Sequences at the 15th European ECML and the 8th European PKDD, 2004.

[7] Claudia Martins Antunes. *Pattern Mining over Nominal Event Sequences using Constraints Relaxations*. PhD thesis, Universidade Tecnica de Lisboa, Instituto Superior Tecnico, 2005.

[8] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *KDD '02: Proceedings of*

the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 429–435, New York, NY, USA, 2002. ACM.

[9] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 223–234, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[10] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359, New York, NY, USA, 2000. ACM.

[11] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. Knowl. Data Eng.*, 16(11):1424–1440, 2004.

[12] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag, 25–29 1996.

[13] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, 42(1-2):31–60, 2001.