GASP: Graph-based Approximate Sequential Pattern Mining for Electronic Health Records

Wenqin Dong[1], Eric W Lee[2][0000−0002−3839−2826], Vicki Stover Hertzberg[2][0000−0002−8834−4363], Roy L Simpson[2], and Joyce C Ho[2][0000−0001−9168−3916]

[1] Carnegie Mellon University
wenqind@andrew.cmu.edu
[2] Emory University
{ewlee4,vhertzb,roy.l.simpson,joyce.c.ho}emory.edu

Abstract. Sequential pattern mining can be used to extract meaningful sequences from electronic health records. However, conventional sequential pattern mining algorithms that discover all frequent sequential patterns can incur a high computational and be susceptible to noise in the observations. Approximate sequential pattern mining techniques have been introduced to address these shortcomings yet, existing approximate methods fail to reflect the true frequent sequential patterns or only target single-item event sequences. Multi-item event sequences are prominent in healthcare as a patient can have multiple interventions for a single visit. To alleviate these issues, we propose GASP, a graph-based approximate sequential pattern mining, that discovers frequent patterns for multi-item event sequences. Our approach compresses the sequential information into a concise graph structure which has computational benefits. The empirical results on two healthcare datasets suggest that GASP outperforms existing approximate models by improving recoverability and extracts better predictive patterns.

Keywords: Sequential Pattern Mining · Healthcare Data

1 Introduction

An increasing amount of electronic healthcare records (EHRs) are collected. Sequential pattern mining (SPM) can help discover important or useful patterns in such data [5] such as the sequence of health interventions that resulted in an unfavorable outcome. Various SPM algorithms have been proposed to discover all the frequent sequential patterns that satisfy a user-defined threshold, or support count (we refer the reader to a survey on the topic [5]). Unfortunately, there are several notable limitations that prevent the widespread usage of these algorithms: computational complexity (in terms of time and memory), generation of noisy frequent patterns, and development for single-item event sequences (or one item per event sequences). However, EHRs are characterized by noisy, multi-item event sequences (i.e., 1 or more items per event sequences). For example, a

Table 1: An example of a sequential database (SDB).

SID	Sequences
1	$\langle\{53, 98\}, \{58, 98\}\rangle$
2	$\langle\{257, 53\}, \{257, 58\}\rangle$
3	$\langle\{10, 53\}, \{257, 259, 58\}, \{98\}\rangle$
4	$\langle\{10\}, \{259, 53, 58\}\rangle$

patient can have multiple treatments for the same visit and the documentation process can be prone to human errors. Thus exact SPMs are not always desirable.

Approximate SPM was proposed to alleviate limitations, by clustering similar sequences together to obtain representative patterns [7,12] or utilizing different data structures such as trees or graphs to approximate the subsequent patterns [1,8] to minimize the number of passes through the data. Unfortunately, there are several limitations of existing approximate SPM algorithms. The majority of the algorithms are developed only for single-item event sequences and are not easily generalizable to multi-item event sequences. Moreover, the empirical results can fail to improve computational efficiency or suffer from poor recall.

We propose a **G**raph-based **A**pproximate **S**equential **P**attern mining algorithm, `GASP`, for multi-item sequential databases (SDBs) constructed from EHRs. Our approach approximates the database as a new weighted graph structure. A sampling-based approach is then utilized to efficiently identify frequent subsequences in the database while providing reasonable recall with the true patterns. Our evaluation showcases that `GASP` requires comparable computational time and memory footprints to state-of-the-art exact SPM methods. Moreover, the approximate patterns contain better predictive power than the exact patterns.

2 PRELIMINARIES

2.1 Notation

Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of unique items (i.e., symbols or alphabets) in the sequential database, SDB. An event or itemset, X, is an unordered collection of items, and denoted as $X = \{i_1, i_2, \ldots, i_k\}$, where i_j is an item from I. A sequence s is an ordered list of itemsets such that $s = \langle X_1, X_2, \ldots, X_n\rangle$. A sequence database contains a list of sequences and is denoted as $SDB = \langle s_1, s_2, \ldots, s_p\rangle$, with p unique sequence identifiers. Table 1 provides an example of SDB which contains four sequences ($p = 4$). A sequence $s_a = \langle a_1, a_2, \ldots, a_m\rangle$ is a subsequence of another sequence $s_b = \langle b_1, b_2, \ldots, b_n\rangle$ if and only if there exist integers $i_1, i_2, \ldots, i_m$ such that $1 \leq i_1 \leq i_2 \leq \ldots \leq i_m \leq n$ and $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_m \subseteq b_{i_m}$. In other words, s_a is contained in s_b. From the first sequence s_1 in Table 1, one potential subsequence is $\langle\{53\}, \{98\}\rangle$.

2.2 Exact Sequential Pattern Mining

Given a sequential database, SDB, the goal of exact SPM is to find all the frequent subsequences (i.e., sequential patterns) that occur in at least some user-specified number of sequences in the SDB. Given the computational challenges of SPM, CM-SPAM and CM-SPADE [11] have been proposed to achieve better time and space scalability by pruning the candidate patterns. Although these algorithms are relatively efficient, they can be susceptible to noise in the data and can fail to deal with long, multi-event sequences.

2.3 Approximate Sequential Pattern Mining

Approximate SPM was proposed to identify "similar" patterns while reducing noise in the patterns and improving computational efficiency. Since patterns may not have a direct one-to-one correspondence to the exact SPMs, minimizing the percentage of dissimilarity between the patterns have been proposed as the objective [12]. Unfortunately, this is limited to single-event patterns and requires specification of the error tolerance. We propose the following approximate SPM framework based on average Levenshtein distance for multi-item SDB.

Definition 1. ***(Levenshtein distance)****. Given two sequences a, b, the Levenshtein distance is the minimum number of single-item edits, including insertions, deletions, and substitutions, required to change a to the b or vice versa.*

$$lev(a,b) = \begin{cases} |a| & \textit{if } |b| = 0 \\ |b| & \textit{if } |a| = 0 \\ lev(tail(a), tail(b)) & \textit{if } a[0] = b[0] \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & \textit{otherwise.} \end{cases} \tag{1}$$

Given a string x, $tail(x)$ refers to a string excluding the first character of x, and starting with the index of 0, $x[n]$ refers to the nth character of x.

Problem Statement. Let s be a frequent subsequence as defined above, and s_1, s_2 be two arbitrary subsequences. s_1 is a better approximation of s than s_2 if $lev(s, s_1) < lev(s, s_2)$. Thus, the goal of approximate SPM is to discover a list of subsequences, $\mathcal{S}_A$, that minimizes the average Levenshtein distance to a pattern in the exact pattern list $\mathcal{S}_E$:

$$\min \frac{1}{|\mathcal{S}_A|} \sum_{s \in \mathcal{S}_A} \min_{s_i \in \mathcal{S}_E} lev(s, s_i). \tag{2}$$

Existing approximate SPM methods can be grouped into two approaches. The clustering approaches, such as ApproxMap [7] and a Hamming Distance-based model [12], mine consensus patterns by grouping the frequent patterns

based on similarity. Yet these algorithms produce poor recall and require additional parameters (i.e., number of clusters). Another area of work tackle online data streams to identify patterns using a single pass of the data such as GraSeq [8], a graph-based approximate SPM algorithm. GraSeq transformed sequences into a directed weighted graph structure with only one scan of data and introduced a non-recursive depth-first search algorithm to acquire approximate sequential patterns. Unfortunately, these works are developed only for single-item sequences and a naïve extension of single-item sequences to multi-item sequences does not yield desirable results (as demonstrated by our empirical results).

3 GASP

We introduce GASP, a graph-based approximate SPM model, to address the limitations of existing approximate SPM algorithms. GASP transforms the SDB into a Markov chain graph, $\mathcal{G}$ and uses a probabilistic generative model to extract the sequential patterns. $\mathcal{G}$ can be viewed as a random sample of the original SDB and thereby retains the same bounds on accuracy of the discovered patterns [10].

3.1 Subsequence Generation

Our graph $\mathcal{G}$ captures the order and relation between all the items I in the SDB. Since the SDB can have multiple items per event, GASP distinguishes between the two scenarios where the two items occur in the same event (type 1), and two items occur in chronological order (type 2).

Definition 2. *(**1-subsequence**). For a sequence s, a 1-subsequence is $y = \langle i_k \rangle$ for all $i_k \in X_1 \cup X_2 \cup ... \cup X_n$.*

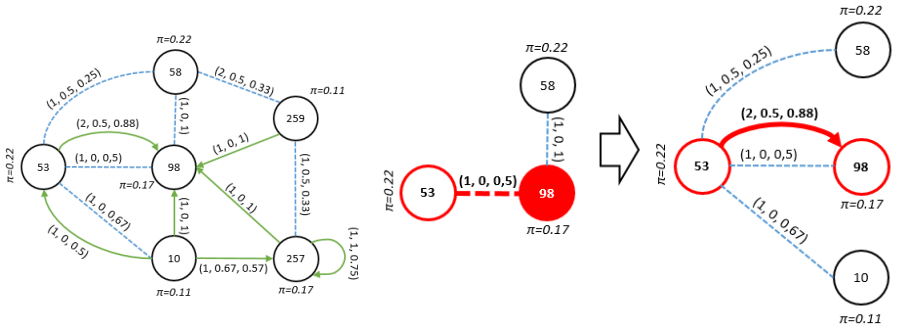
Definition 3. *(**2-subsequence-type-1**). For a sequence s, a 2-subsequence-type-1 is $z^{(1)} = \langle i_k, i_j \rangle$ for all $i_k, i_j \in X_p$ such that $1 \leq p \leq n$.*

Definition 4. *(**2-subsequence-type-2**). For a sequence s, a 2-subsequence-type-2 is $z^{(2)} = \{\langle i_k \rangle, \langle i_j \rangle\}$ for all $i_k \in X_p$, $i_j \in X_q$ and $p < q$.*

GASP scans all the sequences in SDB exactly once to determine all the frequent item sets $\mathcal{Y} = \{y_1, y_2, ...\}$, $\mathcal{Z}^{(1)} = \{z_1^{(1)}, z_2^{(1)}, ...\}$, and $\mathcal{Z}^{(2)} = \{z_1^{(2)}, z_2^{(2)}, ...\}$ and its frequency. As all supersets of infrequent patterns are infrequent, subsequences in $\mathcal{Y}$, $\mathcal{Z}^{(1)}$, $\mathcal{Z}^{(2)}$ that fall below the support count are pruned.

3.2 Graph Construction

GASP constructs a mixed-type graph, $\mathcal{G} = (V, E)$, where V is the set of vertices that represent an item, and E is the set of edges (directed and undirected) to represent the ordering or relation between two items. Each vertex, V_i, corresponds to the i^{th} 1-subsequence in $\mathcal{Y}$. Since items that occur in the SDB are



(a) A example of the partially constructed graph.

(b) A example of random walk to extract $\{\langle 53, 98\rangle, \langle 98\rangle\}$.

Fig. 1: A simplified example of the graph constructed and one iteration of the random walk. Only partial edges are shown in (a). Each node refers to the item in SDB and has a starting item probability (π). A blue dotted undirected edge denotes a type-1 edge and a green directed edge denotes a type-2 edge. Each edge contains the edge weight, event transition weight, and ending probability (w, α, β). For (b), the selection is node 98, type-1 edge with node 53, type-2 edge with node 98 and then terminated to obtain $\{\langle 53, 98\rangle, \langle 98\rangle\}$.

more likely to be part of a frequent pattern, the start probability is set to reflect the likelihood of the item occurring in the SDB: $\pi_i = \frac{freq(y_i)}{\sum_j freq(y_j)}$.

An undirected edge, $(v_i \leftrightarrow v_j)$, represents two items occurring in the same event (or an item in $\mathcal{Z}^{(1)}$). A directed edge, $(v_i \rightarrow v_j)$, denotes the sequential relationship between v_i and v_j, such that v_i occurs in an event prior to v_j (or an item in $\mathcal{Z}^{(1)}$). A weight function, w, is associated with each edge based on the frequency of the particular item set, i.e., $w(v_i \leftrightarrow v_j) = \frac{freq(<v_i, v_j>)}{\sum_\ell freq(<v_i, v_\ell>)}$. The likelihood of staying in the same event is also a function of how many items typically occur in the same event with a specific item.

Proposition 1 *The number of items in an event, X_k with item i_j, is bounded by the maximum number of items in any X_j that contains the item i_j across all the sequences in the SDB, $\langle s_1, s_2, \ldots, s_p\rangle$.*

Given Proposition 1, we introduce a new event transition weight function, α, to capture the likelihood that the next item will be from the same event conditioned on sampling item i_j. Let $|X_k|$ denote the number of items in the event k. For a sequence s, if v_j occurs in the k^{th} event, α_s is defined as:

$$\begin{aligned} \alpha_s(v_i \leftrightarrow v_j) &= \frac{|X_k| - 2}{|X_k| - 2 + \sum_{z=k+1}^{n} |X_z|} \\ \alpha_s(v_i \rightarrow v_j) &= \frac{|X_k| - 1}{|X_k| - 1 + \sum_{z=k+1}^{n} |X_z|} \end{aligned} \tag{3}$$

Then, α is the average weight across all sequences with the item i_j.

Another limitation of existing approximate SPM algorithms is the need to provide a user-defined length for the extracted patterns. We propose to model the length of a candidate pattern as a random variable L. We first introduce two propositions to bound the length of the pattern.

Proposition 2 *The length of a frequent subsequence, ℓ is bounded by the maximum length of all subsequences in the SDB, $\ell \leq \max_{i=1,\ldots,p} |s_i|$.*

Thus the empirical cumulative distribution, $P(L \leq \ell)$, can serve as an upper bound for the maximum number of events in a candidate pattern. Yet, this is independent of the items in the pattern.

Proposition 3 *Given the presence of an item, i_j, in the frequent subsequence, the maximum length of the subsequence is bound by the length of the sequences in the SDB that contain i_j: $\ell \leq \max_{i_j \in s_i, \forall i=1,\ldots,p} |s_i|$.*

Conceptually, if some items occur towards the end of a sequence in the SDB, their presence can be used to terminate the candidate pattern. We introduce a new end weight function, β, to calculate the likelihood that it will terminate the pattern. If v_i, v_j occurs in the sequence s, β_s is defined as:

$$\begin{aligned}\beta_s(v_i \leftrightarrow v_j) &= 1 - \frac{\sum_{z=k+1}^{n} |X_z| + |X_k| - 2}{\sum_{z=1}^{n} |X_z|} \\ \beta_s(v_i \rightarrow v_j) &= 1 - \frac{\sum_{z=k+1}^{n} |X_z| + |X_k| - 1}{\sum_{z=1}^{n} |X_z|}.\end{aligned} \tag{4}$$

The final weight, β is then calculated as the average of the β_s weights across all sequences in the SDB. Figure 1(a) shows the graph for Table 1.

3.3 Random Walk

Random walk was introduced to simulate the likely paths through the graph [9]. Using the same premise, edges, and vertices that have higher weights (or likelihoods) in $\mathcal{G}$, should be traversed more often as they occurred more frequently in the original SDB. To account for the new weight functions and ending probability, `GASP` uses a modified random walk algorithm. The random walk edge weight, d, is determined by the edge weight w and the event transition weight α:

$$d(v_i, v_j) = w(v_i, v_j) \times \alpha(v_i, v_j) \tag{5}$$

The stopping criteria for random walk is also adapted to reflect the number of items currently in the pattern, $\tilde{\ell}$ and the sampled edge, (v_i, v_j). The iteration is stopped based on a Bernoulli random variable

$$L(\tilde{\ell}, (v_i, v_j)) \sim Bernoulli(\frac{1}{2} P(L \leq \tilde{\ell}) + \frac{1}{2} \beta(v_i, v_j)) \tag{6}$$

Algorithm 1 RandomWalk

1: s = Draw v_i randomly using π_i and set $v = v_i$
2: **while** True **do**
3: Calculate the edge weight for all outgoing edges $d(v_i, v_j) = \alpha(v_i, v_j)w(v_i, v_j)$.
4: Choose the new vertex v_j based on edge weight, $d(v_i, v_j)$.
5: Append v_j to the sequence s and set $v = v_j$.
6: Sample pattern end using Eq. (6)
7: **end while**
8: Return candidate pattern s

The detailed steps of our customized random walk are summarized in Algorithm 1 with an example provided in Figure 1(b). Upon the completion of a pattern, the weights of all the edges traversed are summed up to yield the final weight of this particular sequence. These cumulative weights are used for the final candidate pattern ranking.

3.4 Algorithm Complexity

Let P represent the number of sequences in the SDB, N the maximum number of items for a subsequence, I the number of unique items, and L the number of random walk iterations. Since `GASP` only requires a single scan through the SDB, the graph generation has a computational complexity of $O(PN^2)$. For the random walk, the complexity is $O(ILN)$. Hence, the computational complexity of `GASP` is $O(PN^2 + ILN)$. The memory complexity of `GASP` is dominated by the graph and the generated patterns. Only 2-subsequences along with their weight and various probabilities are stored in memory ($O(I^2)$). In the random walk stage, the worst memory scenario is a distinct pattern for each iteration ($O(LN)$). Thus, the memory complexity is $O(I^2 + LN)$.

4 Experiment Setup

4.1 Dataset

We employed two healthcare datasets to assess the performance of `GASP`. CMS is a synthesized and publicly available dataset provided by the Centers for Medicare and Medicaid Services[3]. This dataset contains information about the patients' diagnosis on their visits between the period 2008 to 2009. To construct the SDB, the patient visits are sorted in chronological order and the International Classification of Diseases (ICD-9) billing diagnosis codes are extracted. Clinical Classifications Software (CCS) codes [6] are used to group ICD-9 into broader categories. The Nursing Electronic Learning Lab (NELL) dataset includes electronic health records (EHRs) from Emory Healthcare for type 2 diabetes patients with new onset of cardiovascular disease (CVD) and matched controls. It

[3] `https://www.cms.gov/research-statistics-data-and-systems/Downloadable-Public-Use-Files/SynPUFs/DE_Syn_PUF`

Table 2: Characteristics of each SDB.

Dataset	$\|P\|$	$\|I\|$	Avg $\|s_i\|$	Avg $\|X_k\|$
CMS	68,185	283	40.96	2.22
NELL	12,576	260	12.65	8.55

includes 2,112 cases and 10,464 controls. We extracted diagnosis codes for all patients prior to the CVD index date and group them using CCS codes. The characteristics of the SDBs are summarized in Table 2.

4.2 Experimental Design

All the experiments were run on a single machine, an Amazon EC2 r5.4xlarge instance, with 16 CPU cores and 128GB memory.

4.3 Baseline Methods

We compared `GASP` with the following SPM algorithms: (1) CM-SPAM [11], (2) CM-SPADE [11], (3) GraSeq (fixed), a modified approximate algorithm based on GraSeq [8] to support multi-item event sequences where items in the same event are considered as a single vertex in the graph, (4) GraSeq (variable), an extension of the GraSeq (fixed) to use our proposed random walk algorithm combined with the sequential pattern ending probability to generate variable-length patterns. `GASP`, GraSeq (fixed), and GraSeq (variable) are implemented in Python 3.6. The random walk utilizes multiple threads to further reduce running time on machines with multiple CPUs. The code will be open-sourced in Github upon acceptance[4]. For CM-SPAM and CM-SPADE, we used the SPMF library [4] implementations in Java[5]. FAST [3] and ApproxMap were considered, but the results are omitted due to the poor performance. Other approximate SPM algorithms were not publicly released and thus not compared.

4.4 Evaluation Metrics

We compared the SPM algorithms from multiple perspectives:

- *Computation Time*: The total running time of the algorithm.
- *Memory Usage*: The maximum memory consumed by the algorithm.
- *Levenshtein distance*: The measure defined in Eq. (2).
- *Precision & Recall*: Two measures to capture the relevance of the patterns extracted from the approximate SPM.

$$Prec = \frac{|\mathcal{S}_A \cap \mathcal{S}_E|}{|\mathcal{S}_A|}, \quad Rec = \frac{|\mathcal{S}_A \cap \mathcal{S}_E|}{|\mathcal{S}_E|}$$

[4] `https://github.com/cynthiadwq/GASP`

[5] `https://www.philippe-fournier-viger.com/spmf/`

Table 3: Comparison of SPM algorithms on the two datasets. The memory is reported in megabytes and time is in seconds.

	CMS					NELL				
Model	Time	Mem.	Prec.	Rec.	Lev.	Time	Mem.	Prec.	Rec.	Lev.
CM-SPAM	3798	1937	1.0	0.975	0.034	110	1280	1.0	0.749	0.202
CM-SPADE	815	11008	–	–	–	113	2276	–	–	–
GraSeq (fixed)-5M	304	591	0.106	0.085	1.656	110	481	0.101	0.076	1.697
GraSeq (variable)-5M	311	653	0.147	0.130	1.347	101	352	0.122	0.107	1.458
GASP-5M	322	1036	0.195	0.381	0.527	109	533	0.125	0.255	0.914
GASP-10M	426	1491	0.230	0.507	0.409	219	957	0.172	0.396	0.716

5 Experimental Results

5.1 Pattern Recoverability

CMS The exact SPM algorithms were run using a support threshold of 20% and yielded 127,941 and 124,776 frequent patterns for CM-SPADE and CM-SPAM, respectively. Since CM-SPADE resulted in more patterns, it was used as the ground truth. Table 3 summarizes the performance of the SPM methods on the CMS dataset. GASP can achieve a reasonable approximation of the *exact* frequent patterns generated by CM-SPADE in terms of Levenshtein distance without requiring a trade-off in terms of time or memory. To extract the frequent patterns using CM-SPADE, it uses almost 10× more memory than GASP with 10 M iterations. Moreover, for CM-SPAM, it requires almost 10× the computational time than GASP to produce similar patterns to CM-SPADE. The results also illustrate the importance of variable-length pattern as GraSeq (variable) outperforms GraSeq (fixed) in terms of pattern recoverability. Moreover, GASP (5M) outperforms GraSeq (variable) across all three measures, highlighting the benefit of modeling the different types of two-item subsequences.

NELL The exact SPM algorithms were run using a support threshold of 1% and yielded an average of 1,459,820 and 1,085,201 frequent patterns for CM-SPADE and CM-SPAM, respectively. The results from Table 3 show that GASP-10M is able to identify almost 40% of the original patterns of CM-SPADE, whereas CM-SPAM extracts almost 75% of the original patterns. Moreover, the Levenshtein distance between the original patterns and patterns generated by GASP is less than 1 whereas the two variants of GraSeq have Levenshtein distance greater than 1 and identifies only 10% of the original patterns. While the computation time is similar across CM-SPAM, CM-SPADE, and GASP-5M, GASP-5M requires almost half the memory of CM-SPAM and quarter of the memory of CM-SPADE.

5.2 Pattern Usefulness

We evaluate the usefulness of the extracted patterns as a feature for risk prediction of CVD on NELL. We performed 5 random, stratified 70-30 train-test

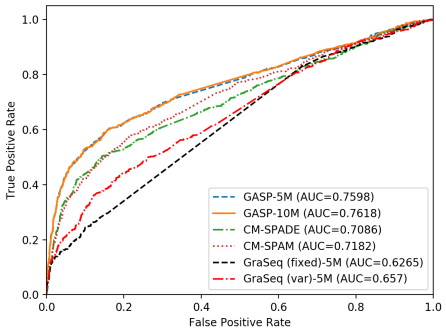


Fig. 2: The ROC curve and AUC score for risk prediction of CVD.

splits where frequent patterns are extracted using the train set, and then the top 500 patterns are used to construct binary features (i.e., the occurrence of the pattern). An XGBoost model [2] is trained and the performance is evaluated using the receiver operating characteristic (ROC) curve and the area under the ROC curve (AUC) shown in Figure 2. `GASP`-5M and `GASP`-10M outperform exact SPM models in terms of AUC. This indicates that exact SPM models identify many noisy patterns while `GASP` generates patterns that are more useful for risk prediction. The results also demonstrate the insensitivity to the specification of the random walk iterations (5 M versus 10M) as there is a limited difference in predictive power. Finally, the results illustrate the impact of approximation to combat the noise inherent in EHRs.

6 Conclusions

In this paper, we propose `GASP`, a new approach for approximate SPM of EHRs. We present a new weighted graph structure using both directed and undirected edges which compresses the sequential information. We also introduce a variant of a random walk model to extract variable-length sequential patterns. Empirical evaluations on two EHR databases suggest that `GASP` reduces the noise in patterns and can enhance pattern usefulness without sacrificing computational and memory efficiency. As approximate SPM is applicable to many other applications, future work can focus on evaluation across multiple domains.

Acknowledgements. This work was supported by the National Science Foundation award IIS-#1838200 and the National Institutes of Health (NIH) awards 1R01LM013323 and 5K01LM012924.

References

1. Chang, J.H., Lee, W.S.: Efficient mining method for retrieving sequential patterns over online data streams. Journal of Information Science **31**(5), 420–432 (2005)

2. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proc. of KDD. pp. 785–794 (2016)
3. Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R.: Fast vertical mining of sequential patterns using co-occurrence information. In: Proc. of PAKDD. pp. 40–52 (2014)
4. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The spmf open-source data mining library version 2. In: Proc. of ECML/PKDD. pp. 36–40 (2016)
5. Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
6. Geraci, J.M., Ashton, C.M., Kuykendall, D.H., Johnson, M.L., Wu, L.: International classification of diseases, 9th revision, clinical modification codes in discharge abstracts are poor measures of complication occurrence in medical inpatients. Medical care pp. 589–602 (1997)
7. Kum, H.C., Pei, J., Wang, W., Duncan, D.: Approxmap: Approximate mining of consensus sequential patterns. In: Proc. of SDM. pp. 311–315 (2003)
8. Li, H., Chen, H.: Graseq: A novel approximate mining approach of sequential patterns over data stream. In: Proc. of ADMA. pp. 401–411 (2007)
9. Pearson, K.: The problem of the random walk. Nature **72**(1867), 342–342 (1905)
10. Raïssi, C., Poncelet, P.: Sampling for sequential pattern mining: From static databases to data streams. In: Proc. of ICDM. pp. 631–636 (2007)
11. Salvemini, E., Fumarola, F., Malerba, D., Han, J.: Fast sequence mining based on sparse id-lists. In: Proc. of ISMIS. pp. 316–325 (2011)
12. Zhu, F., Yan, X., Han, J., Philip, S.Y.: Efficient discovery of frequent approximate sequential patterns. In: Proc. of ICDM. pp. 751–756 (2007)