REEF: Resolving Length Bias in Frequent Sequence Mining

Ariella Richardson
Industrial Engineering
Jerusalem College of Technology
Jerusalem, Israel
Email: richards@jct.ac.il

Gal A. Kaminka and Sarit Kraus
Computer Science
Bar Ilan University
Ramat Gan, Israel
Email: galk,sarit@cs.biu.ac.il

***Abstract*—Classic support based approaches efficiently address frequent sequence mining. However, support based mining has been shown to suffer from a bias towards short sequences. In this paper, we propose a method to resolve this bias when mining the most frequent sequences. In order to resolve the length bias we define *norm-frequency*, based on the statistical z-score of support, and use it to replace support based frequency. Our approach mines the subsequences that are frequent relative to other subsequences of the same length. Unfortunately, naive use of *norm-frequency* hinders mining scalability. Using *norm-frequency* breaks the anti-monotonic property of support, an important part in being able to prune large sets of candidate sequences. We describe a bound that enables pruning to provide scalability. Experimental results on textual and computer user input data establish that we manage to overcome the short sequence bias successfully, and to illustrate the production of meaningful sequences with our mining algorithm.**

Keywords*—*Frequent Sequence Mining; Data Mining; Z-score;

I. Introduction

The frequent sequence mining problem was first introduced by Agrawal and Srikant [1] and by Mannila et al. [2]. There are many possible applications for frequent sequential patterns, such as DNA sequence mining [3], text mining [4] anomaly detection [5] classification [6], and Web mining [7].

Frequent sequential pattern generation is traditionally based on selecting those patterns that appear in a large enough fraction of input-sequences from the database. This measure is known as *support*. In support based mining a threshold termed *minsup* is set. All sequences with a *support* higher than *minsup* are considered frequent.

Support based mining is known to suffer from a bias towards short patterns [8]: Short patterns are inherently more frequent than long patterns. This bias creates a problem, since short patterns are not necessarily the most interesting patterns. Often, short patterns are simply random occurrences of frequent items. The common solution of lowering the *minsup* results in obtaining longer patterns, but generates a large number of useless short sequences as well [9]. Using confidence measures lowers the number of output sequences but still results in short sequences.

Thus, removing the short sequence bias is a key issue in finding meaningful patterns. One possible way to find valuable patterns is to add weights to important items in the data. Yun [10] provides an algorithm for frequent sequence mining using weights. The drawback of this technique is that for many data sets there is no knowledge of what weights to apply. Seno and Karypis [11] propose eliminating the length bias by extracting all patterns with a support that decreases as a function of the pattern length. This solution is based on the assumption that a short pattern must have a very high support to be interesting, and a long pattern may be interesting even with a lower support. Although this is a fair assumption in many scenarios, it is challenging to find a measure that can be used for frequent pattern mining without making an assumption on the relationship between frequency and length. Searching for closed or maximal patterns [12]–[14] is another way to approach this bias. However, mining closed or maximal patterns may not be the best approach to solve the short sequence bias. Using closed and maximal sequences ignores shorter partial sequences that may be of interest. Other approaches include comparing the frequency of a sequence to its subsequences [15], and testing for self sufficient sequences [16]. We propose an algorithm that mines sequences of all lengths without a bias towards long or short sequences. Horman and Kaminka [8] proposed using a normalized support measure for solving the bias. However, their solution is not scalable. Furthermore they cannot handle subsequences that are not continuous or have multiple attributes. We allow holes in the sequence, for example: if the original sequence is ABCD, Horman and Kaminka can find the subsequences AB, ABC, ABCD, BC etc, but cannot mine ACD or ABD, whereas our proposed method can.

In this paper, we present an algorithm for **RE**solving l**E**ngth bias in **F**requent sequence mining (REEF). REEF is an algorithm for mining frequent sequences that normalizes the support of each candidate sequence with a length adjusted z-score. The use of the z-score in REEF eliminates statistical biases towards finding shorter patterns, and contributes to finding meaningful patterns as we will illustrate. However, it challenges the scalability of the approach: z-score normalization lacks the anti-monotonic property used in support based measures, and thus supposedly forces explicit enumeration of every sequence in the database. This renders useless any support based pruning of candidate sequences, the basis for scalable sequence mining algorithms, such as SPADE [17].

In order to provide a means for pruning candidate sequences, we introduce a bound on the z-score of future sequence expansions. The z-score bound enables pruning in the mining process to provide scalability while ensuring closure. Details on how the bound is calculated will be described later in the paper. We use this bound with an enhanced SPADE-like algorithm to efficiently search for sequences with high z-score values, without enumerating all sequences. A previous preliminary study [18] indicates that this bound assists the speedup substantially. We use three text corpora and computer user input to demonstrate how REEF overcomes the bias towards short sequences. We also show that the percentage of real words among the sequences mined by REEF is higher than those mined with SPADE.

The structure of the paper is as follows: Section II provides background and notation and introduces Norm-Frequent Sequence Mining Problem. In Section III the algorithm used for the Norm-Frequent Sequence Mining is described in detail. Experimental evaluation is provided in Section IV, and finally Section V concludes our paper.

II. Norm-Frequent Sequence Mining

Norm-Frequent Sequence Mining solves the short sequence bias present in traditional *Frequent* Sequence Mining. We begin by introducing the notation and the traditional *Frequent* Sequence Mining problem in Section II-A. We then define the *Norm-Frequent* Sequence Mining problem in Section II-B. We explain why the scalability is hindered by the naive implementation of normalized support and how this is resolved in Section II-C. Section II-C addresses scalability by introducing a bound that enables pruning in the candidate generation process. Finally in Section III we bring all parts together to compose the REEF algorithm.

A. Notation and Frequent Sequence Mining

We use the following notation in discussing Norm Frequent Sequence Mining.

event Let $I = \{I_1, I_2, ..., I_m\}$ be the set of all *items*. An *event* (also called an *itemset*) is a non-empty unordered set of *items* denoted as $e = \{i_1, ..., i_n\}$ where $i_j \in I$ is an item. Without loss of generality we assume they are sorted lexicographically. For example, $e = \{ABC\}$ is an event with items A B and C.

sequence A *sequence* is an ordered list of *events*, with a temporal ordering. The sequence $s = e_1 \to e_2 \to ... \to e_q$ is composed of q events. If event e_i occurs before event e_j, we denote it as $e_i < e_j$. e_i and e_j do not have to be consecutive events and no two *events* can occur at the same time. For example, in the sequence s=$\{ABC\} \to \{AE\}$ we may say that $\{ABC\} < \{AE\}$ since $\{ABC\}$ occurs before $\{AE\}$.

sequence size and length The *size* of a sequence is the number of events in a sequence, $size(\{ABC\} \to \{ABD\}) = 2$. The *length* of a sequence is the number of items in a sequence including repeating items. A sequence with length l is called an *l-sequence*. $length(\{ABC\} \to \{ABD\}) = 6$.

subsequence and contain A sequence s_i is a *subsequence* of the sequence s_j, denoted $s_i \preceq s_j$, if $\forall e_k, e_l \in s_i, \exists e_m, e_n \in s_j$ such that $e_k \subseteq e_m$ and $e_l \subseteq e_n$ and if $e_k < e_l$ then $e_m < e_n$. We say that s_j *contains* s_i if $s_i \preceq s_j$. E.g., $\{AB\} \to \{DF\} \preceq \{ABC\} \to \{BF\} \to \{DEF\}$.

database The database D used for sequence mining is composed of a collection of sequences.

support The *support* of a sequence s in database D is the proportion of sequences in D that *contain* s. This is denoted $supp(s, D)$.

This notation allows the description of multivariate sequence problems. The data is sequential in that it is composed of ordered events. The ordering is kept within the subsequences as well. The multivariate property is achieved by events being composed of several items. The notation enables discussion of mining sequences with gaps both in events and in items, as long as the ordering is conserved. The mined sequences are sometimes called patterns.

In traditional support based mining, a user specified minimum support called *minsup* is used to define frequency. A *frequent* sequence is defined as a sequence with a support higher than *minsup*, formally defined as follows:

Definition 1 (Frequent): Given a database D, a sequence s and a minimum support *minsup*. s is *frequent* if $supp(s, D) \geq minsup$.

The problem of frequent sequence mining is described as searching for all the *frequent* sequences in a given database. The formal definition is:

Definition 2 (Frequent Sequence Mining): Given a database D, and a minimum support *minsup*, find all the *frequent* sequences.

In many support based algorithms such as SPADE [17], the mining is performed by generating candidate sequences and evaluating whether they are frequent. In order to obtain a scalable algorithm a pruning is used in the generation process. The pruning is based on the anti-monotonic property of support. This property ensures that support does not grow when expanding a sequence, e.g., $supp(\{AB\} \to \{C\}) \geq supp(\{AB\} \to \{CD\})$. This promises that candidate sequences that are *not frequent* will never generate *frequent* sequences, and therefore can be pruned. *Frequent* sequence mining seems to be a solved problem with a scalable algorithm. However, it suffers from a bias towards mining short subsequences. We provide an algorithm that enables mining subsequences of all lengths.

B. Norm-Frequent Sequence Mining using Z-Score

In this section, we define the problem of *Norm-Frequent* Sequence Mining. We use the statistical z-score for normalization. The z-score for a sequence of length l is defined as follows:

Definition 3 (Z-score): Given a database D and a sequence s. Let $l = len(s)$ be the length of the sequence s. Let μ_l and σ_l be the average support and standard deviation of support for sequences of length l in D. The *z-score* of s denoted $\zeta(s)$ is given by $\zeta(s) = \frac{supp(s) - \mu_l}{\sigma_l}$.

We use the z-score because it normalizes the support measure relative to the sequence length. Traditional mining, where support is used to define frequency, mines sequences that appear often relative to **all** other sequences. This results in short sequences since short sequences always appear more often than long ones. Using the z-score normalization of support for mining finds sequences that are frequent relative to other **sequences of the same length**. This provides an even chance for sequences of all lengths to be found frequent.

Based on the definition of z-score for a sequence we define a sequence as being *Norm-Frequent* if the z-score of the

seq 1:	$\{AB\} \to \{A\}$
seq 2:	$\{AB\} \to \{B\}$
seq 3:	$\{BC\} \to \{A\}$
seq 4:	$\{AB\} \to \{A\}$
seq 5:	$\{BC\} \to \{B\}$
seq 6:	$\{AC\} \to \{B\}$
seq 7:	$\{AB\} \to \{A\}$
seq 8:	$\{AC\} \to \{C\}$
seq 9:	$\{BC\} \to \{C\}$
seq 10:	$\{AC\} \to \{A\}$

Figure 1: Example database

sequence is among the top z-score values for sequences in the database. The formal definition follows:

Definition 4 (Norm-Frequent): Given a database D, a sequence s of length l and an integer k. Let Z be the set of the k highest z-score values for sequences in D, s is *norm-frequent* if $\zeta(s) \in Z$. In other words, we perform top-K mining of the most norm-frequent sequences.

We introduce the problem of *Norm-Frequent* Sequence Mining. This new problem is defined as searching for all the *norm-frequent* sequences in a given database. The formal definition follows and will be addressed in this paper.

Definition 5 (Norm-Frequent Sequence Mining): Given a database D and integer k, find all the *norm-frequent* sequences.

In Figure. 1, we provide a small example. The sequences $\{AB\}$, $\{A\} \to \{A\}$ and $\{B\} \to \{A\}$, of length 2, all have a support of 0.4 and are the most frequent patterns using support to define frequency. Notice that there are several sequences with this support, and no single sequence stands out. Consider the sequence $\{AB\} \to \{A\}$ of length 3. This sequence only has a support of 0.3. However, all other sequences of length 3 have a support no higher than 0.1. Although there are several sequences of length 2 with a higher support than $\{AB\} \to \{A\}$, this sequence is clearly interesting when compared to other sequences of the same length. This example provides motivation for why support may not be a sufficient measure to use. The norm-frequency measure we defined is aimed at finding this type of sequence.

Unfortunately, the z-score normalization test hinders the anti-monotonic property: we **cannot** determine that $\zeta(\{AB\} \to \{C\}) \geq \zeta(\{AB\} \to \{CD\})$.
Therefore, pruning becomes difficult; we cannot be sure that the z-score of a candidate sequence with length l will not improve in extensions of length $l+1$ or in general $l+n$ for some positive n. Therefore, we cannot prune based on z-score and ensure finding all *norm-frequent* sequences. This is a problem since without pruning our search space becomes unscalable.

Another problem with performing *Norm-Frequent* Sequence Mining is that the values for μ_l and σ_l must be obtained for sequences of all lengths prior to the mining process. This imposes multiple passes over the database and hinders scalability.

These important scalability issues are addressed and solved in Section II-C resulting in a scalable frequent sequence mining algorithm that overcomes the short sequence bias.

C. Scaling Up

As we explained in Section II-B, pruning methods such as those described in SPADE [17] cannot be used with *norm-frequent* mining. We propose an innovative solution that solves the scalability problem caused by the inability to prune.

Our solution is to calculate a bound on the z-score of sequences that can be expanded from a given sequence. This bound on the z-score of future expansions of candidate sequences is used for pruning. We define the bound and then explain how it is used. Z-score was defined in definition 3. The bound on z-score is defined in definition 6.

Definition 6 (Z-score-Bound): Given a database D and a sequence s. Let $\mu_{l'}$ and $\sigma_{l'}$ be the average support and standard deviation of support for sequences of length l' in D. The *z-score-bound* of s, for length l' denoted $\zeta^B(s, l')$ is given by $\zeta^B(s, l') = \frac{supp(s) - \mu_{l'}}{\sigma_{l'}}$.

We know that support is anti-monotonic, therefore as the sequence length grows support can only get smaller. Given a candidate sequence s of length l with a support of $supp(s)$ we know that for all sequences s' generated from s with length $l' > l$ the maximal support is $supp(s)$. We can calculate the bound on z-score, $\zeta^B(s, l')$, for all possible extensions of a candidate sequence. Notice that for all sequences s' that are extensions of s, $\zeta(s') \leq \zeta^B(s, l')$. The ability to calculate this bound on possible candidate extensions is the basis for the pruning.

In order to mine *frequent* or *norm-frequent* sequences, candidate sequences are generated and evaluated. In traditional *frequent* sequence mining there is only one evaluation performed on each sequence. If the sequence is found to be *frequent* it is both saved in the list of *frequent* sequences and expanded to generate future candidates, if it is not *frequent* it can be pruned (not saved and not used for generating candidates). For *norm-frequent* mining we perform two evaluations for each sequence. The first is to decide whether the proposed sequence is *norm-frequent*. The second is to determine if it should be expanded to generate more candidate sequences for evaluation. There are two tasks since z-score is not anti-monotonic and a sequence that is not *norm-frequent* may be used to generate *norm-frequent* sequences. This second task is where the bound is used for pruning. The bound on future expansions of the sequences is calculated for all possible lengths. If the bound on the z-score for all possible lengths is lower than the top n z-scores then no possible expansion can ever be *norm-frequent* and the sequence can be safely pruned from the generation process. If for one or more lengths the bound is high enough to be *norm-frequent* we must generate candidates from the sequence and evaluate them in order to determine if they are *norm-frequent* or not. This process guarantees that all *norm-frequent* sequences will be generated.

Using the bound enables pruning of sequences that are guaranteed not to generate *norm-frequent* candidates. The pruning enabled by using the bound resolves the first scalability issue of sequence pruning in the generation process. The second scalability problem of calculating μ_l and σ_l is resolved by calculating the values for μ_l and σ_l on a small sample of the data in a preprocessing stage described below.

III. REEF Algorithm

In this section, we combine all the components we have described in the previous sections and describe the implementation of REEF. The REEF algorithm is composed of several phases. The input to REEF is a database of sequences and an integer *'k'* determining how many Z-scores will be used to find *norm-frequent* sequences. The output of REEF is a set of *norm-frequent* sequences. Initially a sampling phase is performed to obtain input for the later phases. Next we perform the candidate generation phase. First norm-frequent 1-sequences and 2-sequences are generated. Once 2-sequences have been generated, an iterative process of generating candidate sequences is performed. The generated sequences are evaluated, and if found to be *norm-frequent* are placed in the output list of *norm-frequent* sequences. These sequences are also examined in the pruning process of REEF in order to determine if they should be expanded or not.

Sampling Phase - The sampling phase is performed as a preprocessing of the data in order to gather statistics of the average and standard deviation of support for sequences of all possible lengths. This stage uses SPADE [17] with a *minsup* of 0 to enumerate all possible sequences in the sampled data and calculate their support. For each length the support average and standard deviation are calculated. These values are distorted and corrected values are calculated using the technique described in [18]. These corrected values provide the average support μ_l and standard deviation of support σ_l that are used in z-score calculation and the bound calculation.

Candidate Generation Phase - The candidate generation phase is based on SPADE along with important modifications. As in SPADE we first find all 1-sequence and 2-sequence candidates. The next stage of the candidate generation phase involves enumerating candidates and evaluating their frequency.

We make two modifications to SPADE. The first is moving from setting a *minsup* to setting the $'k'$ value. $'k'$ determines the number of z-score values that norm-frequent sequences may have. Note that there may be several sequences with the same z-score value. The reason for this modification is that z-score values are meaningful for comparison within the same database but vary between databases. Therefore, setting the $'k'$ value is of more significance than setting a min-z-score threshold.

The second and major change we make is swapping *frequency* evaluation with *norm-frequency* evaluation. In other words, for each sequence s replace the test of is $supp(s, D) > minsup$ with the test of is $\zeta(s) \in Z$ where Z is the set of the $'k'$ highest z-score values for sequences in D. This replacement of the frequency test with the norm-frequency test is the essence of REEF and our main contribution.

The improved version of sequence enumeration including the pruning is presented in Figure. 2 and replaces the enumeration made in SPADE. The joining of *l*-sequences to generate *l+1*-sequences ($A_i \bigvee A_j$ found in line 6) is performed as in SPADE [17].

Pruning Phase using Bound - Obviously REEF cannot enumerate all possible sequences for norm-frequency evaluation. Furthermore as we discussed in Section II-B the z-score measure is not anti-monotonic and cannot be used for pruning

```
1:  for all x is a prefix in S do
2:      $T_x = \emptyset$
3:  $F_R = \{$k empty sequences$\}$
4:  for all items $A_i \in S$ do
5:      for all items $A_j \in S$, with $j \geq i$ do
6:          $R = A_i \bigvee A_j$ (join $A_i$ with $A_j$)
7:          for all $r \in R$ do
8:              if $\zeta(r) > \zeta($a seq $s$ in $F_R)$ then
9:                  $F_R = F_R \bigcup r \backslash s$ //replace $s$ with $r$
10:             for all $l' = l+1$ to input sequence length do
11:                 if $\zeta^B(r, l') > \zeta($a seq $s$ in $F_R)$ then
12:                     if $A_i$ appears before $A_j$ then
13:                         $T_i = T_i \bigcup r$
14:                     else
15:                         $T_j = T_j \bigcup r$
16:     enumerate-Frequent-Seq-Z-score($T_i$)
17:     $T_i = \emptyset$
```

Figure 2: Enumerate-Frequent-Seq-Z-score(S). Where S is the set of input sequences we are mining for frequent subsequences, A set of *norm-frequent* subsequences is returned, F_R is a list of sequences with the top $'k'$ z-scores

.

while ensuring that norm-frequent candidates are not lost. In Section II-C we introduced the bound on z-score that is used for pruning.

The pruning in REEF calculates $\zeta^B(s, l')$ for all possible lengths $l' > l$ of sequences than could be generated from s. The key to this process that there is no need to actually generate the extensions s' that can be generated from s. It is enough to know the $supp(s)$, μ_l and σ_l for all $l' > l$. If for any length $l' > l$ we find that $\zeta^B(s, l') \in Z$ (in the list of 'k' z-scores) we keep this sequence for candidate generation, if not then we prune it. Using the bound for pruning reduces the search space while ensuring closure or in other words ensuring all frequent sequences are found. The pruning is performed as part of the enumeration described in algorithm Figure. 2. This pruning is the key to providing a **scalable** *norm-frequent* algorithm.

IV. Evaluation

In this section, we present an evaluation of REEF on a corpora of literature of various types. Section IV-A will show that *norm-frequent* mining overcomes the short sequence bias present in *frequent* mining algorithms. In Section IV-B we will provide evidence that the sequences mined with REEF are more meaningful than sequences mined with SPADE.

TEXT is a corpus of literature of various types. We treat the words as sequences with letters as single item events. We removed all formatting and punctuation from text (apart from space characters) resulting in a long sequence of letters. Mining this sequential data for frequent sequences produces sequences of letters that may or may not be real words. The reason we chose to mine text in this fashion is to show how interesting the frequent sequences are in comparison to norm-frequent sequences by testing how many real words are discovered. In other words, we use real words from the text as ground truth against which to evaluate the algorithms. We use three sets of textual data, one is from Lewis Carroll's "Alice's Adventures in Wonderland" [19], another is Shakespeare's "A Midsummer Night's Dream" [20] and the third is a Linux installation guide [21]. Evaluation is performed on segments of the corpus. Each test is performed on five segments.

User **P**attern **D**etection (UPD), is a data set composed of real world data used for evaluation. UPD logs keyboard and mouse activity of users on a computer as sequences, for a detailed description see [18]. Sequences mined from the UPD data can be used to model specific users and applied to security systems as in [22], [23] and [18]. The experiments are run on 11 user sessions.

The input is composed of long sequences. In order to use REEF these sequences are cut into smaller sequences using a sliding window thus creating manageable sequences for mining. The size of the sliding window is termed *input sequence length* in our results. We use a setting of *minsup=1%* and *'k'=50* throughout all experiments and a sample rate of 10% for the preprocessing sampling component. Further details on implementation, running times etc. can be found in [24].

A. Resolving Length Bias in Frequent Sequence Mining

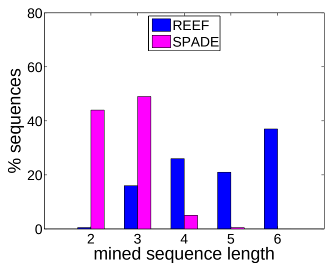
In this section, we establish how REEF successfully overcomes the short sequence bias that is present in the frequent sequence mining techniques. We performed *frequent* sequence mining with SPADE and *norm-frequent* sequence mining with REEF. We compared the lengths of the mined sequences for both algorithms. The results are displayed in Figure. 3. Results are shown for all three TEXT data sets and for the UPD set. The x-axis shows the lengths of the mined sequences. The y-axis displays the percentage of sequences found with the corresponding length. For each possible length we counted the percentage of mined sequences with this length.

The text results on all three text corpora show how SPADE mines mainly short sequences, while REEF manages to mine a broader range of sequence lengths as displayed in Figure. 3(a),(b),(c). REEF results are much closer to known relation between word length to frequency [25] than the SPADE output. In the next section we count how many of these sequences are real words to illustrate superiority of REEF.

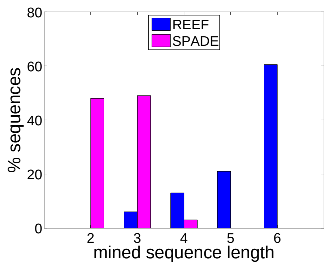
For the UPD data REEF again overcomes the short sequence bias and provides output sequences of all lengths in a more normal distribution than with SPADE. This can be seen in in Figure. 3(d). We must point out that in contrast to the TEXT corpora, there is no known ground truth as to what the length of frequent sequences should be in this domain, and what their distributions are. Thus, there is no way to confirm whether we have found the correct distribution of the frequent sequences. However, we do show that we are not restricted to mining short sequences alone.

B. Mining Meaningful Sequences with REEF

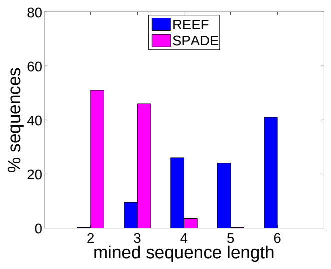
The text domain was chosen specifically in order to illustrate the quality of the output sequences. We wanted a domain where the meaning of interesting sequences was clear. TEXT is obviously a good domain for this purpose since words are clearly more interesting than arbitrary sequences of letters.



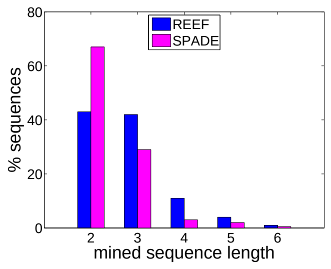
(a) Lewis Carroll



(b) Shakespeare



(c) Linux-Guide

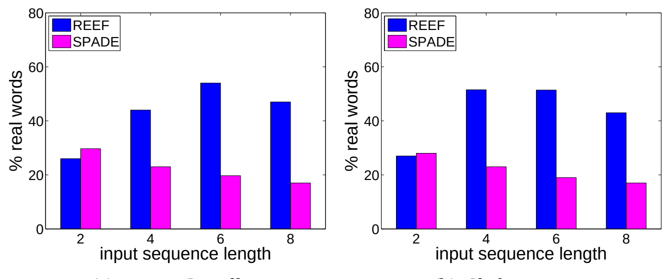


(d) UPD

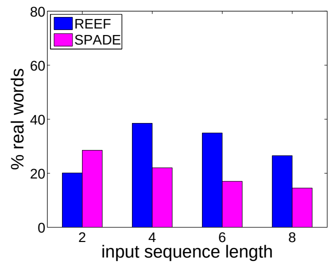
Figure 3: Removal of length bias.

We hope to find more real words when mining text than nonsense words. Our evaluation is performed on three sets of text as described above. Results appear in Figure. 4. We compare results on *frequent* sequence mining using SPADE with *norm-frequent* sequence mining using REEF. The x-axis shows different input sequence lengths (window sizes). For each input sequence length we calculated the percentage of real words that were found in the mined sequences. This is displayed on the y-axis. For example the top 15 mined sequences in Shakespeare using REEF: $\{$*e he,or,e and,her,n th,though,he,s and,her,thee,this,thou,you,love,will*$\}$and using SPADE: $\{$*rth,mh,lr,sf,tin,op,w,fa,ct,ome,ra,yi,em,tes,t l*$\}$ Using REEF yields many more meaningful words than using SPADE.

For all text sets REEF clearly outdoes SPADE by far. REEF



(a) Lewis Carroll (b) Shakespeare



(c) Linux Guide

Figure 4: Percentage of real words found among sequences.

manages to find substantially more words than SPADE for all input lengths. The short input-sequence sizes of 2 does not produce high percentages of real words for REEF or SPADE. Using longer input sequence lengths exhibits the strength of REEF in comparison to SPADE. For input lengths of 4,6 and 8 REEF manages to find a much higher percentage of words than SPADE. Clearly for text REEF performs much better mining than SPADE and the sequences mined are more meaningful.

V. Conclusion and Future Work

We developed an algorithm for frequent sequence mining named REEF that overcomes the short sequence bias present in many mining algorithms. We did this by defining *norm-frequency* and using it to replace support based frequency used in algorithms such as SPADE. In order to ensure scalability of REEF we introduced a bound for pruning in the mining process.

Our experimental results show without doubt that the bias is indeed eliminated. REEF succeeds in finding frequent sequences of various lengths and is not limited to finding short sequences. We illustrated that REEF produces a more variant distribution of output pattern lengths. We also clearly showed on textual data how REEF mines more real words than SPADE. This seems to indicate that when mining sequences are not textual, we can expect to mine meaningful sequences as well. In the future we hope to improve the bound used for mining. Thus providing an algorithm that is more efficient while still producing the high quality sequences we found in REEF.

References

[1] R. Agrawal and R. Srikant, “Mining sequential patterns,” in Proceedings of the Eleventh International Conference on Data Engineering, 1995, pp. 3–14.

[2] H. Mannila, H. Toivonen, and A. I. Verkamo, “Discovering frequent episodes in sequences (extended abstract),” in 1st Conference on Knowledge Discovery and Data Mining, 1995, pp. 210–215.

[3] F. Elloumi and M. Nason, “Searchpattool: a new method for mining the most specific frequent patterns for binding sites with application to prokaryotic dna sequences.” BMC Bioinformatics, vol. 8, 2007, pp. 1–18.

[4] N. Zhong, Y. Li, and S.-T. Wu, “Effective pattern discovery for text mining,” IEEE Transactions on Knowledge and Data Engineering, vol. 24, 2012, pp. 30–44.

[5] W. Fan, M. Miller, S. J. Stolfo, W. Lee, and P. K. Chan, “Using artificial anomalies to detect unknown and known network intrusions,” Knowledge and Information Systems, vol. 6, 2004, pp. 507–527.

[6] C.-H. Lee and V. S. Tseng, “PTCR-Miner: Progressive temporal class rule mining for multivariate temporal data classification,” in IEEE International Conference on Data Mining Workshops, 2010, pp. 25–32.

[7] P. Senkul and S. Salin, “Improving pattern quality in web usage mining by using semantic information,” Knowledge and Information Systems, 2011, pp. 527–541.

[8] Y. Horman and G. A. Kaminka, “Removing biases in unsupervised learning of sequential patterns,” Intelligent Data Analysis, vol. 11, 2007 , pp. 457–480.

[9] C. Luo and S. M. Chung, “A scalable algorithm for mining maximal frequent sequences using sampling,” in ICTAI ’04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence. Washington, DC, USA: IEEE Computer Society, 2004, pp. 156–165.

[10] U. Yun, “An efficient mining of weighted frequent patterns with length decreasing support constraints,” Knowledge-Based Systems, vol. 21, 2008 , pp. 741–752.

[11] M. Seno and G. Karypis, “SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support constraint,” in ICDM ’02: Proceedings of the 2002 IEEE International Conference on Data Mining. Washington, DC, USA: IEEE Computer Society, 2002, p. 418.

[12] P. Tzvetkov, X. Yan, and J. Han, “Tsp: Mining top-k closed sequential patterns,” Knowledge and Information Systems, vol. 7, 2005, pp. 438–457.

[13] C. Luo and S. Chung, “A scalable algorithm for mining maximal frequent sequences using a sample,” Knowledge and Information Systems, vol. 15, 2008, pp. 149–179.

[14] N. Tatti and B. Cule, “Mining closed strict episodes,” in ICDM ’10: Proceedings of the 2010 Tenth IEEE International Conference on Data Mining, 2010, pp. 34–66.

[15] N. Tatti, “Maximum entropy based significance of itemsets,” Knowledge and Information Systems, vol. 17, 2008, pp. 57–77.

[16] G. I. Webb, “Self-sufficient itemsets: An approach to screening potentially interesting associations between items,” ACM Trans. Knowl. Discov. Data, vol. 4, Jan 2010 , pp. 1–20.

[17] M. J. Zaki, “SPADE: An efficient algorithm for mining frequent sequences,” Machine Learning Journal, vol. 42, 2001, pp. 31–60.

[18] A. Richardson, G. Kaminka, and S. Kraus, “CUBS: Multivariate sequence classification using bounded z-score with sampling,” in IEEE International Conference on Data Mining Workshops, 2010, pp. 72–79.

[19] L. Carroll, “Alice’s Adventures in Wonderland,” Project Gutenberg.

[20] W. Shakespeare, “A Midsummer Night’s Dream,” Project Gutenberg.

[21] J. Goerzen and O. Othman, “Debian gnu/linux : Guide to installation and usage,” Project Gutenberg.

[22] A. E. Ahmed, “A new biometric technology based on mouse dynamics,” IEEE Transactions on Dependable and Secure Computing, vol. 4, 2007, pp. 165–179.

[23] R. Janakiraman and T. Sim, “Keystroke dynamics in a general setting,” in Advances in Biometrics, ser. Lecture Notes in Computer Science, vol. 4642/2007. Springer Berlin/ Heidelberg, Aug 2007, pp. 584–593.

[24] A. Richardson, “Mining and classification of multivariate sequential data,” Ph.D. dissertation, Bar Ilan University, 2011.

[25] B. Sigurd, M. Eeg-Olofsson, and J. V. Weijer, “Word length, sentence length and frequency zipf revisited,” Studia Linguistica, vol. 58, 2004 , pp. 37–52.