REEF: Resolving Length Bias in Frequent Sequence Mining Using Sampling

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. Calculation of the bound uses a preprocessing stage on a sample of the dataset. Sampling the data creates a distortion in the samples measures. We present a method to correct this distortion. We conducted experiments on 4 data sets, including synthetic data, textual data, remote control zapping data and computer user input data. Experimental results establish that we manage to overcome the short sequence bias successfully, and to illustrate the production of meaningful sequences with our mining algorithm.**

***Index Terms*—Frequent Sequence Mining; Data Mining; Z-score; Sampling; Multivariate Sequences**

I. Introduction

In a previous study [1] we discussed resolving the length bias in frequent sequence mining. The frequent sequence mining problem was first introduced by Agrawal and Srikant [2] and by Mannila et al. [3]. There are many possible applications for frequent sequential patterns, such as DNA sequence mining [4], text mining [5], anomaly detection [6], [7], classification [8] and Web mining [9].

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 [10]: 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 [11]. 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 [12] 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 [13] 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 [14]–[16] 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 [17], and testing for self sufficient sequences [18]. We propose an algorithm that mines sequences of all lengths without a bias towards long or short sequences. Horman and Kaminka [10] 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 [1], we presented an algorithm for **RE**solving l**E**ngth bias in **F**requent sequence mining (REEF), this algorithm is expanded in the current paper. 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 [19].

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 [20] indicates that this bound assists the speedup substantially, we add more proof of this in the current note. We use three text corpora, input from TV remote control usage 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.

This paper enhances the previous work presented in [1] in several ways. First, we present the method used for the sampling of the data that we use for calculating the bound. We also present an extensive evaluation of the parameter setting for this method using the various data sets. Second, we report the runtime results for use of the bound and discuss them. Finally, we present the results of our experimental evaluation on two extra data sets, TV remote control usage (Zapping) and a synthetic data set, that were not reported in [1].

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

II. Related Work

The topic of frequent sequence mining is highly researched. This essential data mining task has broad applications in many domains and is used for a variety of applications such as agent modeling. In multi-agent settings there are many usages to modeling agents. A group of coordinating agents must have a clear model of each agent in the group and agents working in adversarial environments must be able to model their opponent. In cases where there is no prior behavior library (as is often the case) it is necessary to use the observed behavior in order to deduce the model. This task can be performed using the multivariate sequences generated by agents, mining them for frequent patterns and using them for modeling. An example to this type of application is presented by Kaminka et al. [21] where the RoboCup soccer simulation games are used to model the various team behaviors.

One of the prominent applications of frequent sequence mining is classification of a sequential dataset. In [20], the behavior of people in a predefined group is observed, and frequent patterns are mined. These patterns are used in for classifying a given behavior as belonging to a specific person. The same application has been applied to a commercial setting in the personalized television domain. The work we will present is an extension of the mining component described in [20].

Support based algorithms for frequent sequence mining were first introduced by Agrawal and Srikant [2], where the algorithms AprioriAll, AprioriSome and DynamicSome were introduced. These algorithms naturally expand frequent itemset mining to frequent sequence mining. Itemsets do not contain a sequential ordering, whereas sequences do. The algorithms perform pattern mining in sequences of itemsets (events) and find frequent patterns in the input. The itemsets typically contain multiple items. Later they introduced the more efficient GSP [22] which has been broadly implemented and used since.

Since the search space for these mining problems is incredibly large other support based algorithms were introduced to improve the speed and efficiency of the mining process. SPADE [19], introduced by Zaki, is an algorithm for frequent sequence mining that belongs to the family of support based mining algorithms. SPADE outperforms GSP, due to the use of a vertical layout for the database and a lattice-theoretic approach for search space decomposition. We adopt the method presented in SPADE [19] and adapt it to use a normalized support for finding frequent sequences.

The key idea in many of these support based algorithms is the generation of candidate sequences. The candidate sequences are subsequences of the input-sequences in the database. *Frequent* candidate sequences are both placed in the set of mined *frequent* sequences, as well as used to generate the next generation of candidates. First, 2-sequences (sequences of length 2) are generated, then they are used to create 3-sequences etc: pairs of l-sequences with common prefixes are combined, to create an $l+1$-sequence.

Generating all possible candidate sequences is infeasible and results in an unscalable solution. Therefore, a pruning is introduced to this process. Candidates that are *not frequent* are pruned. They are not used to generate the next generation of candidates. The reason this can be done is based on the anti-monotonic property of support. Support has a nice anti-monotonic property promising that it does not grow when a candidate sequence is expanded. This promises that candidate sequences that are *not frequent* will never generate *frequent* sequences, and therefore can be pruned. Thus, the anti-monotonic property is very important and ensures scalability of the mining.

Alongside the rich variety of support based mining algorithms Mannila et al. [3] proposed an algorithm for mining frequent episodes, a type of frequent sequence, in an input composed of a single long sequence. Frequent *episode* mining algorithms find frequent items that are frequent within a single sequence whereas frequent support based sequence mining searches for items that reoccur in multiple sequences. Tatti and Cule [16] proposed mining closed episodes that are represented as DAGs. This algorithm cannot handle multivariate sequences. Salam and Khayal [23] introduced a method for mining top-k frequent patterns without the use of a minimum support. They generate patterns of length 2, and then use a top-

down mechanism that only generates the top maximal frequent itemsets. They build a graphical representation of the data and search for maximal cycles in the graph.

The problem of Frequent Sequence Mining has been solved with many algorithms, an extended survey can be found in [24]. Often, the *frequent* sequences found are often insufficient. Unfortunately, support based mining methods suffer from a bias towards shorter sequences as has been shown in [10]. This means that in the frequent sequence mining, short sequences are found more often than long sequences. This is very problematic since these short sequences are often not very interesting as we will illustrate in Section V-E.

Several attempts have been made to address this bias. One possibility is to force large patterns by searching for closed patterns as in TSP [14] or maximal patterns such as MSPS [15]. 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. We propose an algorithm that mines sequences of all lengths without a bias towards long or short sequences.

In LPMiner [25] (itemset mining) and SLPMiner [13] (sequence mining) Seno and Karypis introduce a length decreasing support constraint in order to overcome the short sequence bias. This is based on the observance that an interesting short sequence must be very frequent (have a very high support) to be interesting. Long sequences on the other hand may be interesting with a lower support. SLPMiner is a heuristic approach whereas in our work we attempt to find a general solution based on support normalization.

An alternative approach is taken by Yun and Legget in WSpan [12]. They introduce a weighted mining algorithm, for sequences with weighted items. Using weights in the mining process is very useful since it provides more input than using frequency alone. Unfortunately, this is of no assistance in domains where there is no information on what weights to apply. Our solution requires no knowledge on what weights should be used and can be implemented in any domain.

The methods for solving the bias towards short subsequences suggested in [12], [13], [25] are heuristic. They are based on forcing long sequences to be mined. In contrast, Horman and Kaminka [10] proposed using a statistical normalization of support. The support measure is normalized in relation to sequence length. They showed how support normalization enables finding frequent subsequences with different lengths in an unbiased fashion. Using normalized support makes no assumptions on the relation between length to support, or on the relative weights of the items in a database as were made in the other methods.

Normalization for frequent pattern mining has been performed in the past. SEARCHPATTOOL [4] uses z-score for normalization of mined frequent patterns. It first performs the sequence mining using a support based algorithm and then selects the significant sequences using the z-score measure. In [26] z-score is used to normalize the data in preprocessing stage, before any mining is performed. We also use the z-score measure for normalization. We show how to use the z-score measure for scale-up in the mining task. To the best of our knowledge this application of z-score is novel, and has been applied only in [10].

Although Horman and Kaminka [10] successfully solve the statistical bias using normalization, their method suffers from three problems. The first difficulty with the method proposed by Horman and Kaminka, solved in this paper involves the scalability of the algorithm. Using the normalized support ruins the anti-monotonic property used for pruning in support based mining. Unfortunately, this makes pruning impossible and therefore the algorithm is unscalable. The second difference between this paper their work is that as opposed to [10] where the mined sequences must be continuous in the original sequence, we allow holes in the sequence. An example would be if the original sequence is ABCD, the previous method can find the subsequences AB, ABC, ABCD, BC etc, but cannot mine ACD or ABD, whereas our algorithm mines both types. The third difference is that the previous method could not handle multiple attributes, as opposed to our approach that can.

With the scalability spoiled it seems there is a need to choose between a scalable algorithm to one that can fully overcome the short sequence bias. In this paper, we propose an algorithm that can do both. The algorithm we present uses normalized support to overcome the short sequence bias successfully while using a pruning method with a sampling unit to solve scalability issues.

III. 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 III-A. We then define the *Norm-Frequent* Sequence Mining problem in Section III-B. We explain why the scalability is hindered by the naive implementation of normalized support and how this is resolved in Section III-C. Section III-C addresses scalability by introducing a bound that enables pruning in the candidate generation process and Section III-D describes the Sampling component. Finally, in Section IV, 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 $s = e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_q$ 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=$\{ABC\} \rightarrow \{AE\}$ 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, $size(\{ABC\} \rightarrow \{ABD\}) = 2$. 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*. $length(\{ABC\} \rightarrow \{ABD\}) = 6$.

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., $\{AB\} \rightarrow \{DF\} \preceq \{ABC\} \rightarrow \{BF\} \rightarrow \{DEF\}$.

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 [19], 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., $supp(\{AB\} \rightarrow \{C\}) \geq supp(\{AB\} \rightarrow \{CD\})$. 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 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 $\{AB\}$, $\{A\} \rightarrow \{A\}$ and $\{B\} \rightarrow \{A\}$, 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 $\{AB\} \rightarrow \{A\}$ 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 $\{AB\} \rightarrow \{A\}$, 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 $\zeta(\{AB\} \rightarrow \{C\}) \geq \zeta(\{AB\} \rightarrow \{CD\})$.
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 $l + n$

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

Figure 1: Example database.

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 III-C resulting in a scalable frequent sequence mining algorithm that overcomes the short sequence bias.

C. Scaling Up

As we explained in Section III-B, pruning methods such as those described in SPADE [19] 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.

D. Sampling for Norm-Frequent Mining

Norm-frequent mining uses the z-score defined in Definition 3 and the bound described in Definition 6. Both these measures make use of the average and standard deviation of support for each subsequence length (μ_l and σ_l). We must calculate these values prior to the sequence mining. The naive way to calculate these values would be to generate all possible subsequences and calculate these measures. However, this is obviously irrelevant as making a full expansion completely defeats the purpose of mining with the z-score pruning.

Therefore, we propose extracting a small sample of the database and calculating these values on the sample. For the sample, full expansion is feasible and generates the necessary measures while ensuring scalability.

However, there is a problem that arises with the sampled measures. They do not reflect the full database measures correctly. It has been shown by [11], [15], [27], [28], that there is a distortion, also termed overestimation, in the values of support calculated on a sample of a database relative to support calculated over a full database. Similarly, the average and standard deviation of support suffer a distortion in the sampled data.

1) Effects of Sampling Distortion: We use Figure 2 to demonstrate how the distortion affects sequence mining. The norm-freq sequences that are mined using z-score and calculated with statistics from the full database are displayed in column 1. The top most stripes (light) represent the most *norm-frequent* sequences and the bottom (dark) represent sequences that are *not norm-frequent* (rare). Column 3 shows the *norm-frequent* sequences discovered using averages and

standard deviations for z-score calculation from the sampled database, the colors match the coloring in the full database, the ordering is based on sampled results. One notices that the order is confused and rare sequences in dark greys show up relatively high in the list. *Norm-frequent* (light) sequences are pushed down as rare. The black stripes at the side of the column represent sequences that did not appear at all in the *norm-frequent* list when using the full data set and appeared when using the sampling. It is obvious that sequences are shifting around and *norm-frequent* sequences are being chosen as rare and vice versa. Therefore, the distortion badly affects sequence mining. In column 2 we use the correction displayed in the next section and improve this shifting. The rare sequences show up further down with the correction than without, as do the candidates that didn't appear in the original list. Although this is just an example on one small set of data it conveys the effects of the distortion and the correction.

2) Chernoff Bounds and Hoeffding Inequalities: We would like to evaluate how far off the sampled statistics are from the real statistics. One might suggest using Chernoff bounds as in [27], [29] or Hoeffding inequalities as in [28] for this task. In [27]–[29], the aim is to show how far off sampled support is from real support for a single subsequence. The appearance of a subsequence in each sequence in the sample is described as a random variable with a Bernoulli distribution. These random variables are **independent**, and the Chernoff bounds or Hoeffding inequalities can be used. The scenario we are using is different. Instead of looking at the accuracy of the support on the sampled data we are looking at the **average** and **standard deviation** of support for a subsequence of a specific length. Unlike the application of Chernoff and Hoeffding bounds in [27]–[29], where the random variable was independent, the random variable in our setting is the average of support of sequences for a given length. This random variable is strongly **dependent** and therefore the known bounds are problematic to apply. There are also situations where although Chernoff bounds can be applied, it is problematic to apply them because a very large sample of the database is needed, as in [27]. For cases where Chernoff and Hoeffding bounds cannot be applied, or situations where one chooses not to apply them we propose a method of distortion correction.

3) Sampling Distortion Correction Method: We introduce a method for correcting the distortion that can be used for any data set. This method finds the model of the distortion for various input sequence lengths and sample rates using the non-linear regression function *nlsfit()* in the R Project for Statistical Computing [30]. Once we have modeled the distortion, correcting it is immediate. The model provides an equation that determines the exact distortion value of average support or standard deviation for a given input sequence length and sample rate. A simple inverse multiplication provides the corrected value.

In order to perform the regression we must propose functions and then perform the non-linear regression to set the parameters. We list the equations we propose using based on our experimental experience as described in Section V-B.

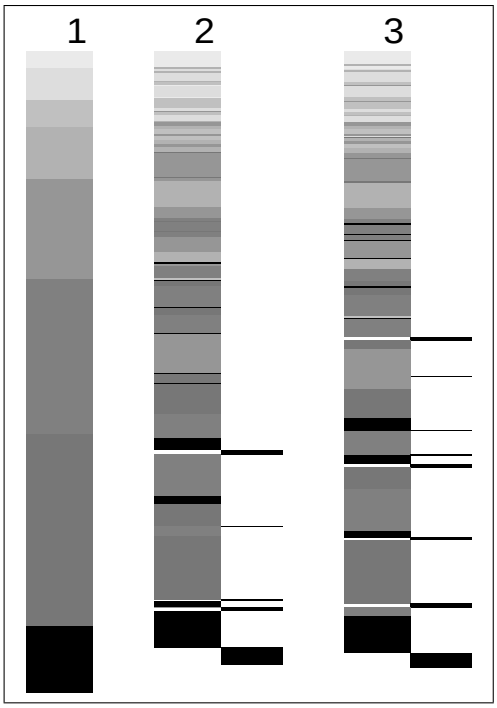


Figure 2: Sampling distortion effect.

The variables in the equations are len (length of the input-sequences) and smp (sampling rate). The coefficients that are determined in the non-linear regression are a, b, c, d. The functions we propose using are:

$$a \times (len-1)^{b \times log(smp)} + c \times smp \quad (1)$$

$$a \times (len)^{b \times smp + c} + d \times smp \quad (2)$$

$$a \times (len-1)^{b \times smp + c} + d \times smp \quad (3)$$

$$(len-1)^{a} + e \times smp^{b} + c \times len + d \times smp \quad (4)$$

For the standard deviation of support we perform distortion correction in a similar fashion using the following equations for approximation.

$$a * (smp^{b}) \quad (5)$$

$$(smp^{b}) + a \quad (6)$$

A tool such as the R Project for Statistical Computing [30] is used to find the correct parameters for these equations as demonstrated in detail in Section V-B.

Once we have found the equations that represent the distortion for average support and standard deviation of support, for a certain type of data set, correction of this distortion is simple. For new data in these sets we can select any sample rate and calculate the distortion correction of average and standard deviation for each possible sequence length. We multiply the sampled values by the inverse of the distortion and use the results as the average and standard deviation of support in the z-score calculation for *norm-frequent* mining.

We found the proposed equations to be general and provide good approximations for different data sets, as shown in

Section V-B, and therefore suggest they can be used for other data sets as well. However, for data sets where these equations do not provide good approximations the same method we used can be applied while using different equations.

Now that we have presented the full method for sampling the data set and calculating the values for μ_l and σ_l we have a complete scalable algorithm that mines *norm-frequent* sequences without a bias to short sequences, Section IV puts all the pieces together and describes the full algorithm.

IV. 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 [19] 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 Section III-D. 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.

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

Figure 3: 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.

The improved version of sequence enumeration including the pruning is presented in Figure 3 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 [19].

Pruning Phase using Bound - Obviously REEF cannot enumerate all possible sequences for norm-frequency evaluation. Furthermore, as we discussed in Section III-B, the z-score measure is not anti-monotonic and cannot be used for pruning while ensuring that norm-frequent candidates are not lost. In Section III-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 3. This pruning is the key to providing a **scalable** *norm-frequent* algorithm.

V. EVALUATION

In this section, we present an evaluation of REEF on several data sets, described in Section V-A. We first demonstrate how to use our sampling distortion method in Section V-B. Next, in Section V-C we compare runtime of the algorithms and justify the use of the bound that was introduced in Section IV. Then in Section V-D will show that *norm-frequent* mining overcomes the short sequence bias present in *frequent* mining algorithms. In Section V-E, we will provide evidence that the sequences mined with REEF are more meaningful than sequences mined with SPADE.

A. Data Sets and Experimental Settings

The evaluation is performed on 4 data sets. One of these is a synthetic data set, three use real world sequential data.

Syn is the synthetic data generated with the IBM QUEST Synthetic Data Generator [31]. QUEST generates data for various data mining tasks, including frequent sequence mining. We generated sequences with the following parameters: Number of customers in database = 1000, Average number of intervals per sequence = 3, Average number of transactions per interval = 3, Number of items = 10, all other settings are the default settings. The tests in the evaluation are performed on 5 synthetic sets with these parameters.

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" [32], another is Shakespeare's "A Midsummer Night's Dream" [33] and the third is a Linux installation guide [34]. Evaluation is performed on segments of the corpus. Each test is performed on five segments.

UPD: **U**ser **P**attern **D**etection, 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. Sequences mined from the UPD data can be used to model specific users and applied to security systems as in [35], [36] and [20]. The experiments are run on 11 user sessions. The data is collected throughout the whole work session and not just at login. Each activity is logged along with the time and date it occurs. The data is then converted into the following events: pressing a key, time between key presses, key-name, mouse click, mouse double click, time between mouse movements. For each session the events are saved in sequences.

Zapping is composed of data that we gathered on remote control usage. In each household members were asked to identify themselves as they begin watching TV, by pressing a designated button on the remote, and then the "zapping sequence" is saved, in other words the buttons they pressed on the remote while they were watching. This sequence is converted into the following events: Button pressed, Time passed since last activity and Time of day. This interesting data set in the domain of personalized television learns personal usage patterns to provide personal services as in [37] and [20]. Each zapping session generates a single long sequence. Evaluation is performed on 10 sets.

For all these data sets 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. The comparison made between REEF to SPADE is delicate since SPADE uses *minsup* to define how many sequences to mine whereas REEF uses *'best'* as described in Section IV. Adjusting these settings changes the runtime and may change the quality and lengths of the mined sequences. Although these parameters are similar in nature they cannot be set to be exactly the same for experiments. We consistently use a single setting of *minsup=1%* and *'k'=50* throughout all experiments and a sample rate of 10% for the preprocessing sampling component.

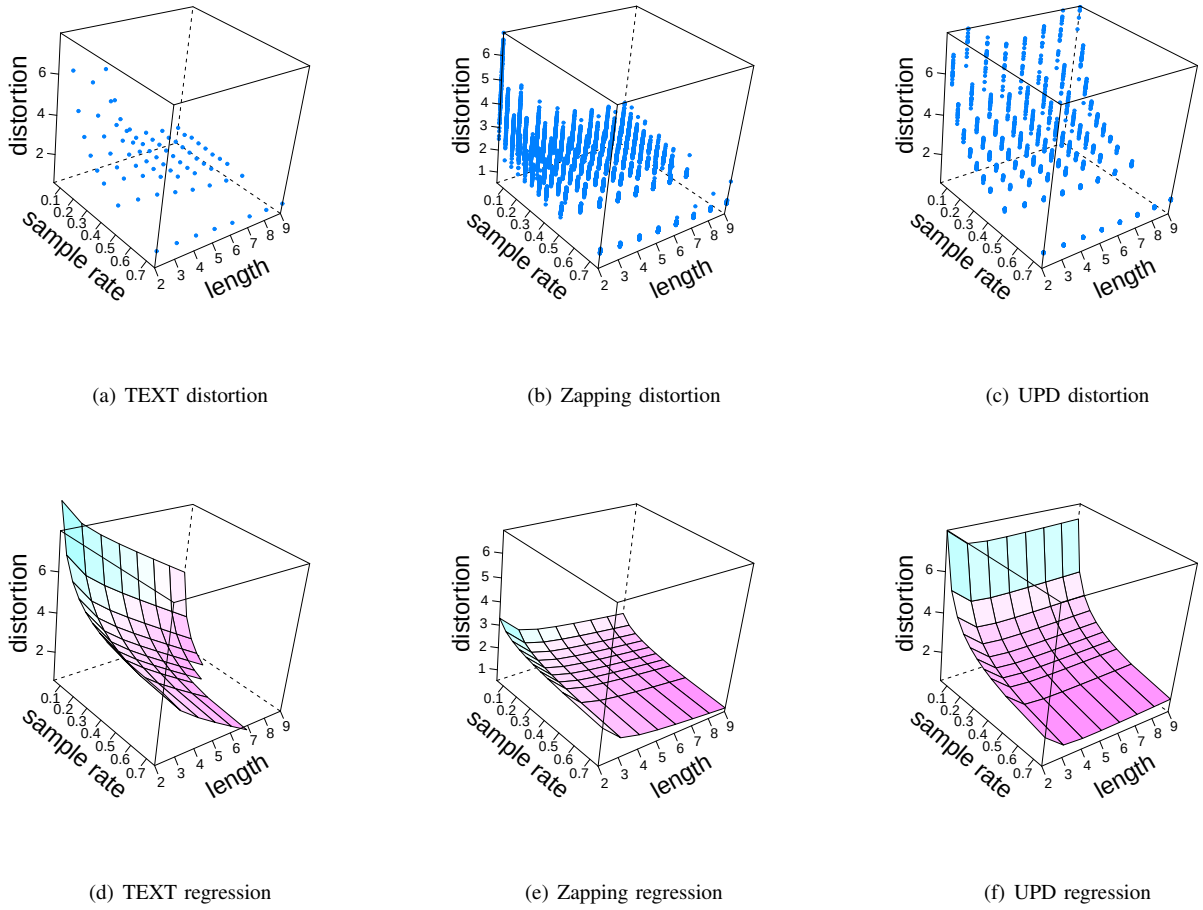
B. Sampling Distortion Correction

We will demonstrate how to perform the distortion correction described in Section III-D3, for several data sets. We found a single equation to model distortion for all the data sets we investigated. Although this does not imply that the same model fits all possible data sets, it is a strong indication that this may be the case. For data sets where this does not hold, the same method we used can be applied to find other models. The data sets we used are the TEXT data set the Zapping data set and the UPD dataset (described in the previous section).

Figure 4 (a),(b),(c) displays the distortion ratio between sampled average support to full data average support on all three data sets. The data used for this analysis is excluded from the experimental evaluation performed in Sections V-C, V-D and V-E. We used approximately half the data for this analysis and half for the experimental evaluation. Each point is an instance of the dataset. The distortion is calculated on each instance for various input sequence lengths and sample rates. The distortion obviously has an orderly structure that we want to find.

We modeled the distortion using non-linear regression. We used R Project for Statistical Computing [30] in order to find a general formula for calculating the correction factor. We need two correction parameters: one for the average support, the other for the standard deviation of support.

We first describe the average support correction. We noticed that when we set the sample rate, the distortion ratio follows a nonlinear function of the length, shown in Figure 5. On the other hand, if we set the length, then the distortion ratio follows a nonlinear function of the sample rate, shown in Figure 6. Therefore, the distortion of average support is dependent both on length and on sample rate and we are



(a) TEXT distortion (b) Zapping distortion (c) UPD distortion

(d) TEXT regression (e) Zapping regression (f) UPD regression

Figure 4: Distortion ratios of average support on sampled data in (a), (b) and(c). Regression surfaces of equation 4 in (d),(e) and (f).

looking for a function $f(len, smp)$ where *len* is the length of a sequence and *smp* is the sample rate.

In previous research [20], we investigated the distortion on the Zapping data alone. We tried to build a combination of the power and logarithmic functions that we saw when looking at each variable, into a single function. This led us to investigating Equations (1), (2) and (3) in Section III-D3. However, when we tried performing regression for other data sets we discovered that for UPD these were not the best candidates, and did not even converge on the TEXT data. We suspect over-fitting of the regression on the Zapping data. Realizing that the shape of the distortion is reminiscent of a stretched paraboloid we tried regression with Equation (4) in Section III-D3 and found that this best suits all three data sets and was therefore selected as the distortion model. The regression surfaces for Equation (4) in Section III-D3 appear in Figure 4 (d),(e),(f). Values of the parameters for non linear least of squares regression appear in Table I.

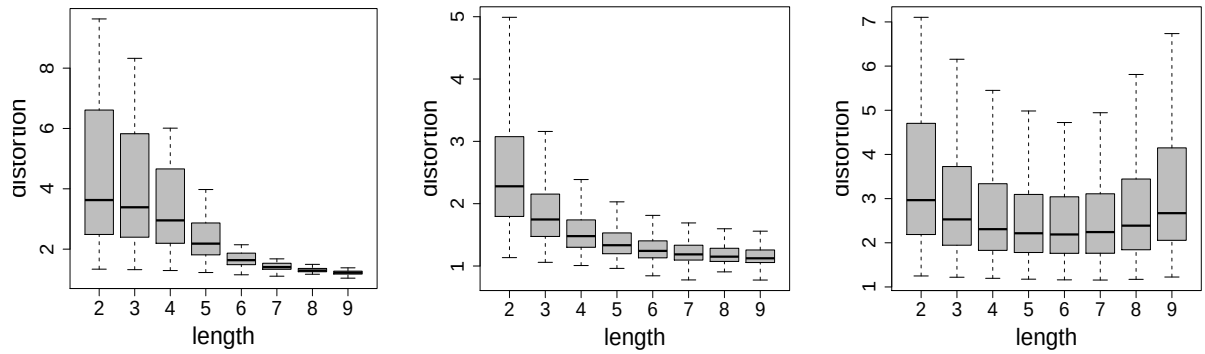
Standard deviation of distortion is linear relative to length (see Figure 7), and is a nonlinear function of sample rate (see Figure 8). Therefore, the only variable involved is the sample rate. The sampling distortion correction we found for zapping in [20] fits the UPD and TEXT data as well. The equations we tested are Equations (5) and (6) in Section III-D3, we chose Equation (5). Regression parameters appear in Table II.

num	func	RSE	a	b
Zapping	5	2	0.928601	-0.799307
Zapping	6	2	-0.101002	-0.776648
UPD	5	0.6055	1.027204	-0.835350
UPD	6	0.6057	0.043373	-0.843472
TEXT	5	2.053	1.059	-0.768

TABLE II: Regression parameter values for standard deviation of support.

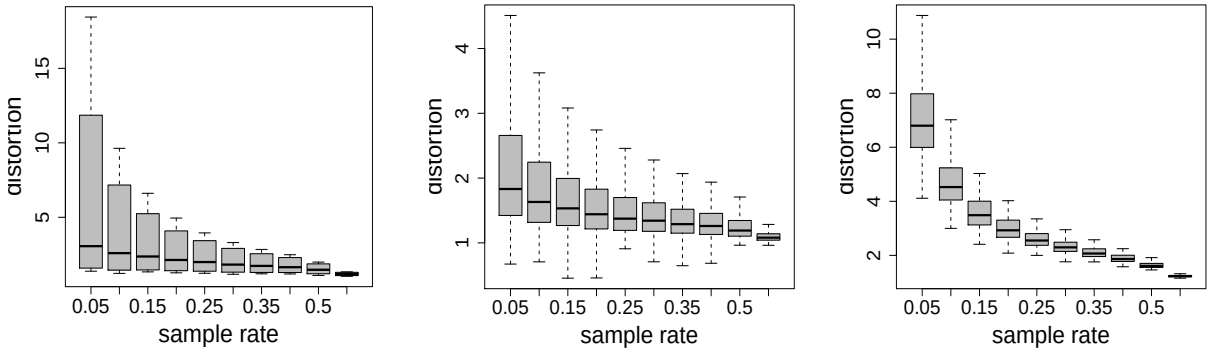
C. REEF Runtime and Bound Pruning

We show how the use of the bound enables speedup. Although the main focus of REEF is not speed but rather output quality, we show that REEFs' runtime is comparable with existing algorithms. We compare the runtime for two versions of REEF to SPADE. REEF refers to the full algorithm described in Section IV. NB-REEF refers to the same



(a) TEXT (b) Zapping (c) UPD

Figure 5: Length cross cut of distortion ratio for average support.



(a) TEXT (b) Zapping (c) UPD

Figure 6: Sample rate cross cut of distortion ratio for average support.

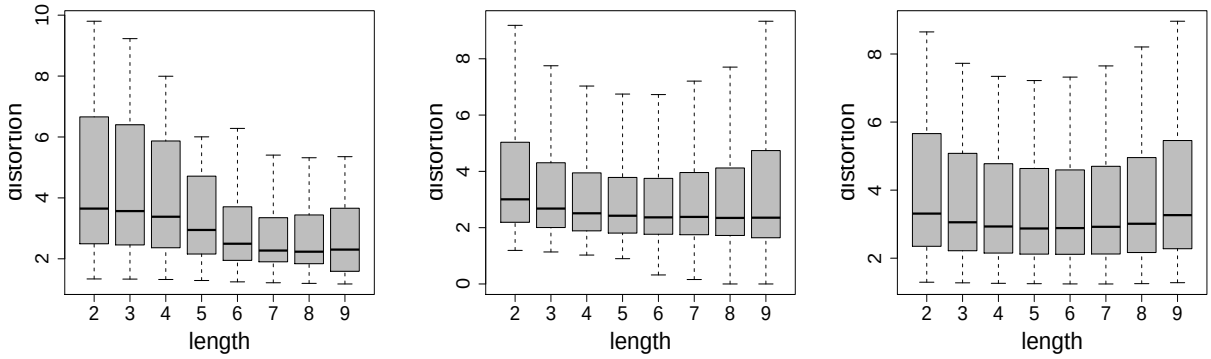
algorithm but without the use of the bound, or in other words without pruning. The runtime includes both the sampling time and the actual mining time for both REEF and NB-REEF. We compare REEF in two versions, one with the use of the bound and the other without. We added SPADE for completeness. We knew in advance that SPADE is faster than our algorithm, but displaying runtime for SPADE provides an order of magnitude for the bound comparison. Results for all data sets appear in Figure 9. There are four types of data sets as described in Section V-A, however the TEXT dataset is composed of three sets of textual data, thus in the results in Figure 9 there are 6 graphs. The x-axis represents input-sequence length. For the synthetic data we had full control over input sequence length and thus present results for all values. For the real data sets the input sequence length is controlled by the number of attributes in an event. This results in varying values along the x-axis for the results. The y-axis displays runtime of the algorithm in seconds. We tested the runtime for various input-sequence lengths. Each point on the graph is the average of five runs.

The first important observation to make is the importance of the pruning bound. For all data sets the pruning noticeably reduces runtime and is an important component of REEF. This is particularly noticeable on the synthetic data in Figure 9(d), UPD data in Figure 9(e) and in the Zapping data in Figure 9(f). This difference grows with input sequence length and becomes more important as input length grows.

The other important result is that the REEF runtime is comparable with that of SPADE. Although SPADE is faster than REEF they are close in runtime. The reason SPADE is often faster than REEF is because *minsup* provides a tighter pruning bound than the one we use in REEF. However, faster may not be better. The tight pruning results in the creation of short sequences. In the next section, we show that there is a tradeoff between runtime to the length of mined sequences, and show how REEF although slightly slower than SPADE has better performance. By overcoming the short sequence bias REEF produces a better distributed set of mined sequences.

data set	func	RSE	a	b	c	d	e
Zapping	1	0.3975	3.561935	0.183471	-3.438119		
Zapping	2	0.3596	5.109377	0.751144	-0.528501	-5.712356	
Zapping	3	0.3391	3.789649	0.703963	-0.465705	-4.124942	
Zapping	4	0.3998	-1.664660	-0.229604	-0.067215	-0.087310	1.226132
UPD	1	1.21	4.935761	-0.009716	-6.576244		
UPD	2	1.102	8.852121	0.648309	-0.302588	-15.3049	
UPD	3	1.141	6.932194	0.476740	-0.220498	-10.9230	
UPD	4	0.5916	-2.417082	-0.665367	0.037497	-0.392893	0.928968
TEXT	4	1.899	-1.416	-0.409	-0.559	1.387	2.719

TABLE I: Regression parameter values for average support.



(a) TEXT (b) Zapping (c) UPD

Figure 7: Length cross cut of distortion ratio for standard deviation.

D. Resolving Length Bias in Frequent Sequence Mining

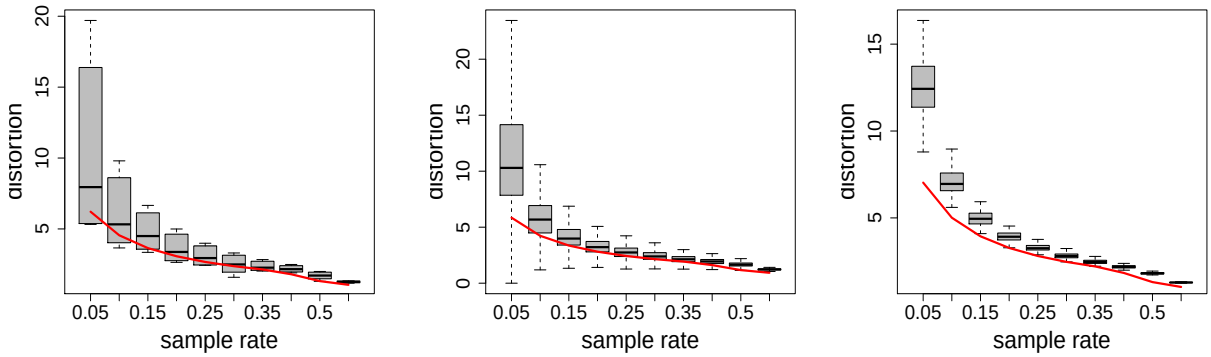
In this section, we establish how REEF successfully overcomes the short sequence bias that is present in the frequent sequence mining techniques. We performed *frequent* sequence mining with SPADE and *norm-frequent* sequence mining with REEF. We compared the lengths of the mined sequences for both algorithms. The results are displayed in Figure 10. Results are shown for the Syn, UPD, Zapping and three TEXT data sets. 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 synthetic data set in Figure 10(d) displays the clearest description of the algorithmic behavior. While SPADE outputs mainly sequences with a length of 2, some with a length of 3, very little with a length of 4 and no longer sequences, REEF outputs sequences with lengths varying from 2 to 6 and with a bell shaped distribution. REEF captures the real nature of the synthetic data and the correct distribution of sequence length.

In the TEXT data set, the 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 10(a),(c),(b). REEF results are much closer to known relation between word length to frequency [38] than the SPADE output. In the next section, we count how many of these sequences are words to illustrate superiority of REEF.

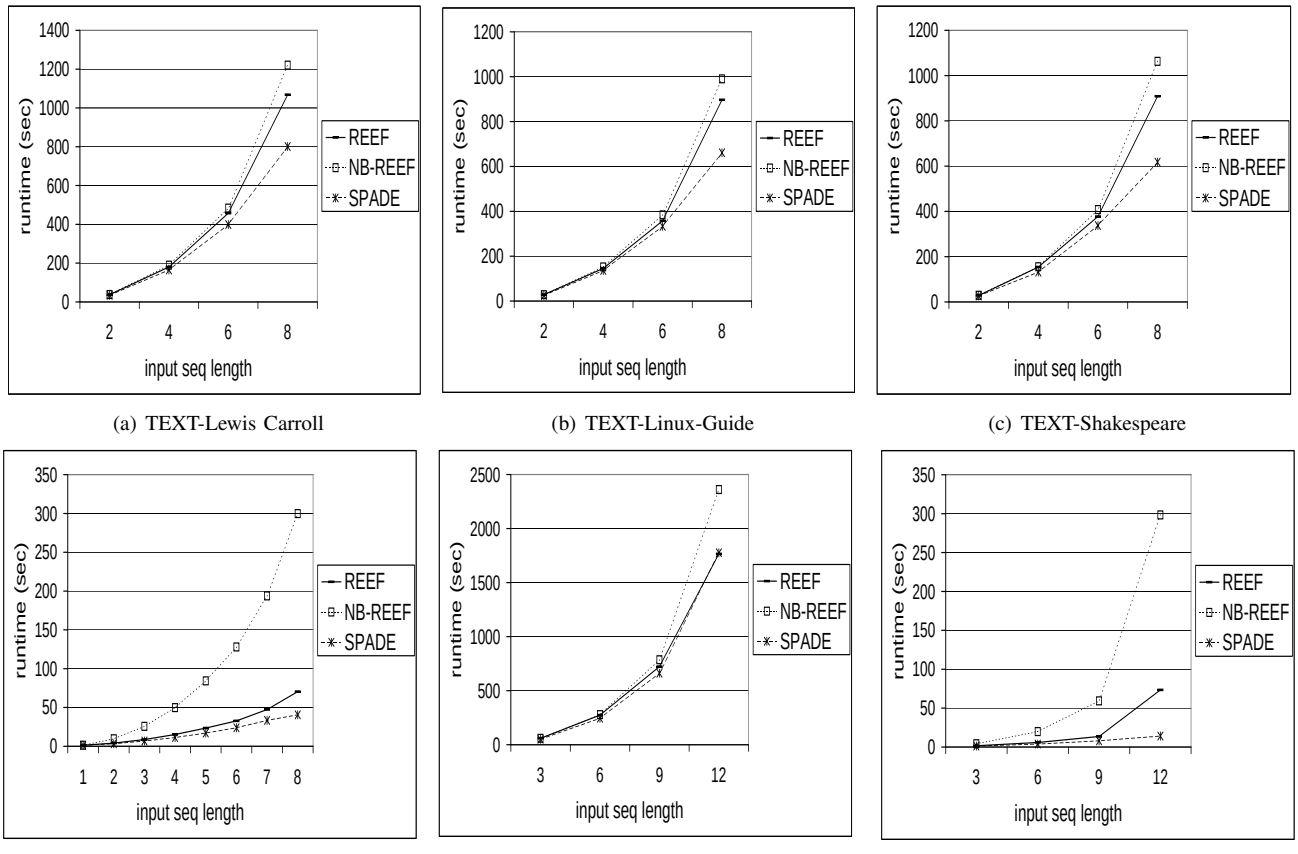
For the Zapping and UPD data REEF again overcomes the short sequence bias and provides output sequences of all lengths in a more normal distribution than with SPADE. This can be seen in Figure 10(e). Note that in contrast to the TEXT corpora, there is no known ground truth as to what the length of frequent 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. However, we do show that we are not restricted to mining short sequences.

An interesting data set is the Zapping set. Although REEF allows for fair mining of all lengths the sequences found both with REEF and with SPADE are short, and there are no sequences with lengths higher than 3 as shown in Figure 10(f). This seems to imply that the frequent sequences in this set really are short. For this data set it would be more beneficial to use SPADE than REEF since there is not much quality to be gained from the slightly longer runtime with REEF. The Zapping set is different to all other three sets, where the extra runtime is clearly worthwhile, since the output sequences tend to be better representatives of the data set. Results on all four sets clearly show the tradeoff in the mining algorithms between time to sequence quality. Frequent sequence mining in support based algorithms such as SPADE generate short frequent sequences quickly. In contrast, norm-frequent mining such as the one we presented in REEF takes slightly longer, but generates sequences with a broader length distribution as we show in Section V-E.



(a) TEXT (b) Zapping (c) UPD

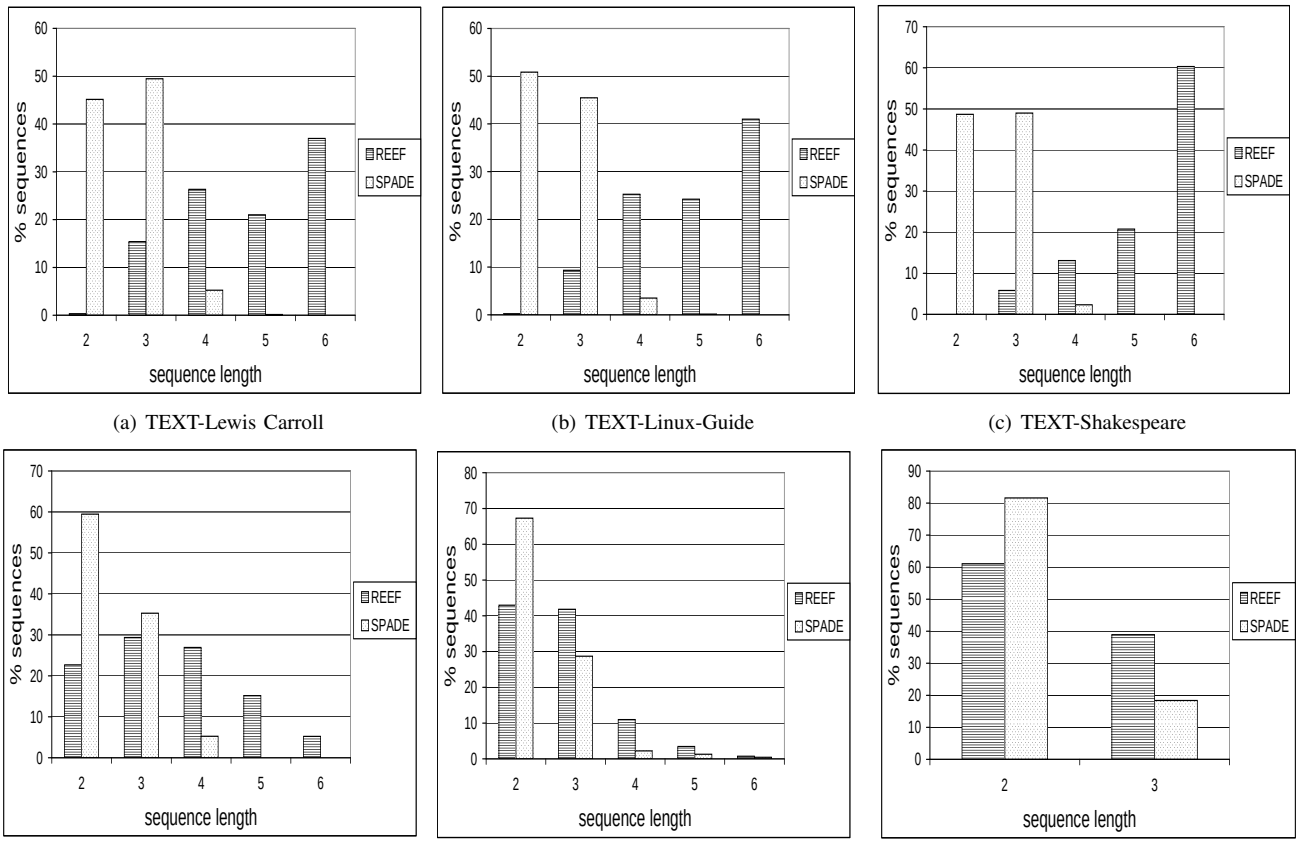
Figure 8: Sample rate cross cut of distortion ratio for standard deviation.



(a) TEXT-Lewis Carroll (b) TEXT-Linux-Guide (c) TEXT-Shakespeare

(d) Synthetic (e) UPD (f) Zapping

Figure 9: Runtime. Comparing REEF (with bound), NB-REEF (without bound) and SPADE.



(a) TEXT-Lewis Carroll (b) TEXT-Linux-Guide (c) TEXT-Shakespeare

(d) Synthetic (e) UPD (f) Zapping

Figure 10: Removal of length bias.

E. 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. 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 11. 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.

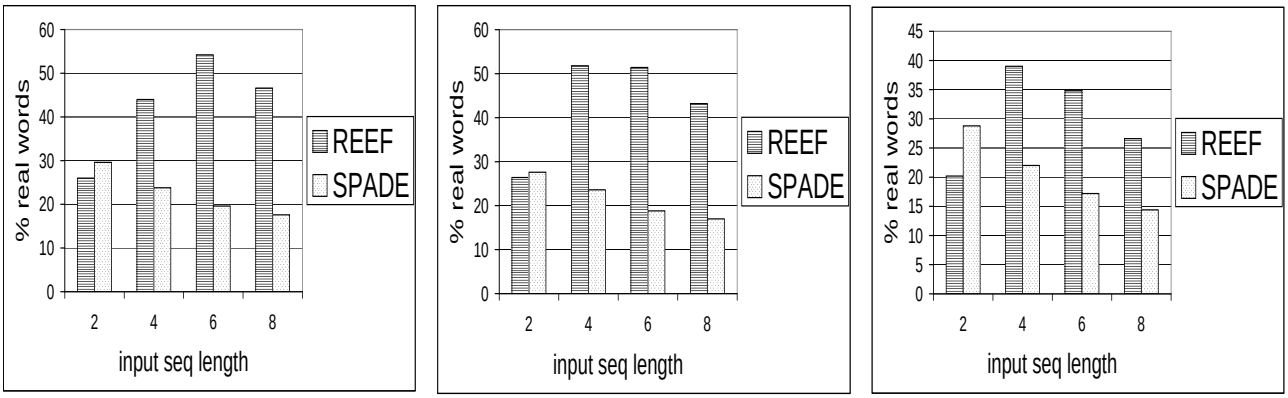
For all text sets REEF clearly outdoes SPADE by far. REEF manages to find substantially more words than SPADE for all input lengths. The short input-sequence sizes of 2 does not produce high percentages of real words for REEF or SPADE. Using longer input sequence lengths exhibits the strength of REEF 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. Although the runtime for SPADE was shorter than for REEF the tradeoff between runtime and output quality is clearly illustrated on the textual data. For many data sets, as for TEXT, it is worth spending more time to the more meaningful sequences in the mining process.

VI. 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. This makes the runtime for REEF comparable to that of SPADE.

The use of the bound requires a preprocessing stage to calculate statistics on a sample of the data set. As this sampling creates a distortion in the sampled measures, we present a method to correct this distortion.

Our extensive experimental evaluation is performed on four different types of data sets. They are a mixture of synthetic



(a) Lewis Carroll (b) Shakespeare (c) Linux Guide

Figure 11: Percentage of real words found among sequences.

and various real world data sets, thus providing a broad performance analysis of REEF. 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 show the scalability of REEF and addressed the tradeoff between runtime to quality of mined 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. Although REEF requires slightly longer runtime than SPADE the nature of the mined sequences makes this worthwhile. 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] A. Richardson, G. A. Kaminka, and S. Kraus, "Reef: Resolving length bias in frequent sequence mining," in *IMMM 2013, The Third International Conference on Advances in Information Mining and Management*, 2013, pp. 91–96.

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

[3] 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.

[4] 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, pp. 1–18, 2007.

[5] N. Zhong, Y. Li, and S.-T. Wu, "Effective pattern discovery for text mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 1, pp. 30–44, 2012.

[6] 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, no. 5, pp. 507–527, 2004.

[7] T. Lane and C. E. Brodley, "An application of machine learning to anomaly detection," in *Proc. 20th NIST-NCSC National Information Systems Security Conference*, 1997, pp. 366–380.

[8] 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.

[9] P. Senkul and S. Salin, "Improving pattern quality in web usage mining by using semantic information," *Knowledge and Information Systems*, no. 3, pp. 527–541, 2011.

[10] Y. Horman and G. A. Kaminka, "Removing biases in unsupervised learning of sequential patterns," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 457–480, 2007.

[11] 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.

[12] U. Yun, "An efficient mining of weighted frequent patterns with length decreasing support constraints," *Knowledge-Based Systems*, vol. 21, no. 8, pp. 741–752, 2008.

[13] 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.

[14] P. Tzvetkov, X. Yan, and J. Han, "Tsp: Mining top-k closed sequential patterns," *Knowledge and Information Systems*, vol. 7, no. 4, pp. 438–457, 2005. [Online]. Available: http://dx.doi.org/10.1007/s10115-004-0175-4

[15] C. Luo and S. Chung, "A scalable algorithm for mining maximal frequent sequences using a sample," *Knowledge and Information Systems*, vol. 15, no. 2, pp. 149–179, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10115-006-0056-0

[16] N. Tatti and B. Cule, "Mining closed strict episodes," in *ICDM '10: Proceedings of the 2010 Tenth IEEE International Conference on Data Mining*, no. 1, 2010, pp. 34–66.

[17] N. Tatti, "Maximum entropy based significance of itemsets," *Knowledge and Information Systems*, vol. 17, no. 1, pp. 57–77, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10115-008-0128-4

[18] G. I. Webb, "Self-sufficient itemsets: An approach to screening potentially interesting associations between items," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 1, pp. 1–20, jan 2010. [Online]. Available: http://doi.acm.org/10.1145/1644873.1644876

[19] M. J. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," *Machine Learning Journal*, vol. 42, no. 1/2, pp. 31–60, 2001.

[20] 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.

[21] G. A. Kaminka, M. Fidanboylu, A. Chang, and M. M. Veloso, "Learning the sequential behavior of teams from observations," in *RoboCup 2002: Robot Soccer World Cup VI*, ser. LNAI, G. A. Kaminka, P. U. Lima, and R. Rojas, Eds. Berlin: Springer-Verlag, 2003, no. 2752, pp. 111–125.

[22] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *EDBT '96 Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, 1996, pp. 3–17.

[23] A. Salam and M. S. H. Khayal, "Mining top-k frequent patterns without minimum support threshold," *Knowledge and Information Systems*, 2011.

[24] C. H. Mooney and J. F. Roddick, "Sequential pattern mining – approaches and algorithms," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 19:1–19:39, Mar. 2013. [Online]. Available: http://doi.acm.org/10.1145/2431211.2431218

[25] M. Seno and G. Karypis, "LPMiner: An algorithm for finding frequent itemsets using length-decreasing support constraint," in *ICDM '01 Proceedings of the 2001 IEEE International Conference on Data Mining* , 2001.

[26] M. Bilenko, S. Basu, and M. Sahami, "Adaptive product normalization: Using online learning for record linkage in comparison shopping," in *Proceedings of the Fifth IEEE International Conference on Data Mining*, ser. ICDM '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 58–65.

[27] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara, "Evaluation of sampling for data mining of association rules," in *RIDE '97 Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE '97) High Performance Database Management for Large-Scale Applications*, 1997, pp. 42–50.

[28] C. Raissi and P. Poncelet, "Sampling for sequential pattern mining: From static databases to data streams," *ICDM '07: Proceedings of the 2007 IEEE International Conference on Data Mining*, vol. 0, pp. 631–636, 2007.

[29] H. Toivonen, "Sampling large databases for association rules," in *Proceedings of the 22th International Conference on Very Large Data Bases*, ser. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 134–145. [Online]. Available: http://portal.acm.org/citation.cfm?id=645922.673325

[30] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[31] R. Agrawal, M. Mehta, J. Shafer, and R. Srikant, "The QUEST data mining system," in *Proc. of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, 1996, pp. 244–249.

[32] L. Carroll, "Alice's Adventures in Wonderland," Project Gutenberg. [Online]. Available: www.gutenberg.org/ebooks/11

[33] W. Shakespeare, "A Midsummer Night's Dream," Project Gutenberg. [Online]. Available: //www.gutenberg.org/ebooks/2242

[34] J. Goerzen and O. Othman, "Debian gnu/linux : Guide to installation and usage," Project Gutenberg. [Online]. Available: www.gutenberg.org/ebooks/6527

[35] A. E. Ahmed, "A new biometric technology based on mouse dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, pp. 165–179, 2007.

[36] 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, August 30 2007, pp. 584–593.

[37] C. Earl, S. Patrick, and P. I. A. Lloyd, "Fuzzy logic based viewer itentification," Patent No. WO/2007/131069, 2007.

[38] B. Sigurd, M. Eeg-Olofsson, and J. V. Weijer, "Word length, sentence length and frequency zipf revisited," *Studia Linguistica*, vol. 58, no. 1, pp. 37–52, 2004. [Online]. Available: http://dx.doi.org/10.1111/j.0039-3193.2004.00109.x