International Journal of Advanced Research in

Computer Science Engineering and Information Technology

Volume: 4 Issue: 3 22-Mar-2015,ISSN_NO: 2321-3337

Proficient Pattern Discovery in Sequence Data Sets

Yogesh S Lonkar

Asst. Professor, Department of Computer Science & Engineering, Karmayogi Engineering College, Shelve-Pandharpur, India.

.

ABSTRACT—*Available sequence mining algorithms mostly focus on subsequence mining. In this mining biological DNA and protein motif mining, require proficient mining of "estimated" patterns that are adjacent. The few available algorithms that can be useful to find such adjacent estimated pattern mining have disadvantage like poor scalability, lack of guarantee in finding the pattern, and difficult in adapting to other applications. In this paper, we present a new algorithm called FLexible and Accurate Motif DEtector (FLAME). FLAME is a flexible suffix-tree-based algorithm that can be used to find frequent patterns with a variety of definitions of motif (pattern) models. It is also correct, as it always finds the pattern if it live. Using both actual and artificial data sets, we show that FLAME is rapid, scalable existing algorithms on a variety of performance metrics. In addition, based on FLAME, we also address a more general problem, named complete structured motif extraction, which allows mining recurrent combinations of motifs under comfortable limitation.*

KEYWORDS— Motif, sequence mining, suffix tree.

1,INTRODUCTION

In a number of sequential data mining applications, the goal is to discover frequently occurring patterns. To illustrate the characteristics of such an operation, consider Figure 1. This figure shows the percentage change in the stock price for a company over the previous minute's average price, for several minutes in a day. An interesting mining question on this dataset is: "Are there any frequently recurring patterns in this time series dataset?" Finding such patterns in stock price data can provide valuable insights that inform trading strategies. In Figure 1, the bold segments highlight a pattern that occurs four times in the dataset. Note that the recurring subsequences are similar, but not identical. The challenge in discovering such patterns is to allow for some *noise* in the matching process. At the heart of such a method is the definition of a pattern, and the definition of similarity between two patterns. This definition of similarity can vary from one application to another. A simple approach in the case of data such as in Figure 1 is to define a tolerance value, /, and consider two sequences to be similar if the corresponding numerical values in the sequences are within / of each other.

This approximate subsequence mining problem is of particular importance in computational biology, where the challenge is to detect short sequences, usually of length 6-15, that occur frequently in a given set of DNA or protein sequences. These short sequences can provide clues regarding the locations of so called “regulatory regions", which are important repeated patterns along the biological sequence. The repeated occurrences of these short sequences are not always identical, and some copies of these sequences may differ from others in a few positions. The similarity metric that is used here could be complex - for example, when comparing proteins, a similarity matrix like PAM [1] or BLOSUM [2], may be used for comparing the "distance" between each symbol (protein) pair. These frequently occurring patterns are called *motifs* in computational biology. In the rest of this paper, we use this term to describe *frequently occurring approximate sequences*.

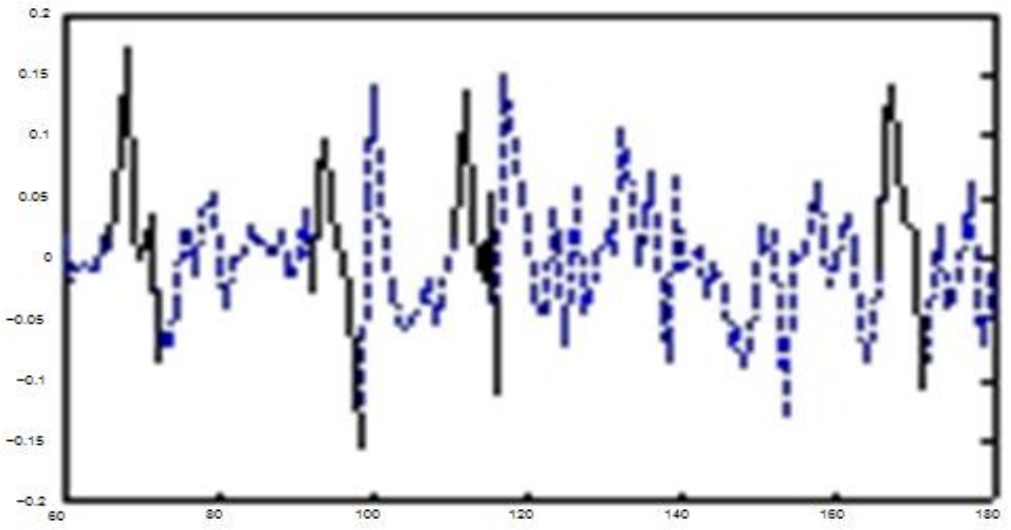

Fig 1. Stock Data: A frequent approximate pattern is highlighted in bold.

Clearly, different applications require different similarity models to suit the kind of noise that they deal with. It is desirable for a motif mining algorithm to be able to deal with a variety of notions of similarity. In this paper, we present a powerful new model for approximate motif mining that fits several applications with varying notions of approximate similarity, including the examples described above. We also present FLAME (**FL**exible and **A**ccurate **M**otif D**E**tector) - a novel motif mining algorithm which can efficiently find motifs that satisfy our model.

We note that the problem of motif mining is related to the problem of mining for frequent itemsets [3], and frequent subsequences [4]. The problem of finding frequently occurring (non-contiguous) subsequences in large sequence databases has been extensively studied in previous works [4]- [8]. Traditionally, B is called a subsequence of A, if B can be constructed by projecting out some of the elements of sequence A. For instance, if A is the sequence "a,b,a,c,b,a,c", the sequence "a,b,b,c" is a subsequence constructed by choos- ing the 1 , 2 , 5 , and 7^{th} elements from the original sequence and omitting the rest. While mining for frequent *non-contiguous* subsequences has many uses, it is not appropriate for many applications such as DNA and protein motif mining. A subsequence constructed by gluing together distant parts of the original sequence is not meaningful in these applications. In mining for motifs, we are interested in *contiguous* subsequences. Furthermore, previous work on non-contiguous subsequence models cannot easily incorporate noise tolerance in the way that contiguous motif models can. In short, subsequence mining and motif mining are different data mining operations, and there are distinct applications of each of these. This paper focuses on the contiguous subse- quence (motif) mining problem. Readers closely familiar with traditional (non-contiguous) subsequence mining algorithms may note that some of these methods can be adapted to mine for contiguous subsequences (e.g. [6], [7], [9], [10]). In the extended version of this paper [11], we compare our method with some of these methods, and show that FLAME is faster by an order of magnitude.

Motivated by the problem of finding frequent patterns in DNA sequences, which has profound importance in life sciences, the computational biology community has developed numerous algorithms for detecting frequent motifs using the Hamming distance notion of similarity. YMF [12], Weeder [13], MITRA [14], and Random Projections [15] are examples of algorithms in this category. Compared to this class of algorithms, we show that FLAME is more flexible, and can use more powerful match models. We also demonstrate through empirical evaluation that FLAME is more scalable than these existing methods and can be an order of magnitude faster for mining large databases.

There are several applications of motif mining in addition to those mentioned above. It is often the first step in discovering association rules in sequence data ("basic shapes" in [16] and "frequent patterns" in [17]). It can also be used to find good seeds for clustering sequence datasets [18]. Records of medical signals, like ECG or respiratory data [19] from patients can also be mined to find signals that can indicate a potentially critical condition.

We make the following contributions in this paper:

1) We present a powerful new model that is very general and applicable in many emerging applications. We demonstrate the power and flexibility of this model by applying it to datasets from several real applications.

2) We describe a novel motif mining algorithm called FLAME (**FL**exible and **A**ccurate **M**otif D**E**tector) that uses a concurrent traversal of two suffix trees to efficiently explore the space of all motifs.

The remainder of the paper is organized as follows: Section II presents related work, and Section III describes our model for motifs. In Section IV, we present the FLAME algorithm. Section VI contains our experimental results, and Section VII contains our conclusions.

2,RELATED WORK

There is a vast amount of literature on mining databases for frequent patterns [21]-[23]. Early work focused on mining association rules [3]. The problem of mining for subsequences was introduced in [4]. Subsequence mining has several applications, and many algorithms like SPADE [5], BIDE [6], CloSpan [7] (and several others) have been proposed as improvements over [4]. Yang et al. [8] use a statistical sampling based method with a compatibility matrix to find patterns in the presence of noise. However, they primarily focus on subsequence mining, while we focus on contiguous patterns.

Some subsequence mining algorithms allow certain constraints. Constraints which limit the maximum gap between two items in the subsequence make it possible to use these algorithms to mine for contiguous patterns. Algorithms like EXMOTIF [24] and RISO [25] are designed to efficiently find multi-motifs, i.e. Simple motif patterns separated by variable length spaces. FLAME does not target the multi-motif problem, but can be be used as a building block for multi- motif mining. Algorithms such as cSPADE [9], CloSpan [7], Pei et al. [10], [26] can be adapted to mine for exact contiguous motifs. An obvious reason why these are unsuitable for approximate frequent pattern mining is that these algorithms do not include a notion of noise or an approximate match. Furthermore, they tend to be inefficient even when used for exact substring mining. FLAME, on the other hand is extremely efficient even for approximate substrings.

Several other algorithms such as the Yeast Motif Finder [12] (YMF), Weeder [13], MITRA [14] have been used for finding motifs. YMF is a simple algorithm that computes the statistical significance of each motif. YMF scales very poorly with increasing complexity of motifs, and thus cannot be easily adapted to other applications. Weeder is a suffix tree based algorithm that makes certain assumptions about the way the mismatches in an instance of the motif are distributed. This makes Weeder extremely fast, but it is not guaranteed to always find the motif. Weeder too, cannot be adapted for other motif models. MITRA is a mismatch tree based algorithm which

uses clever heuristics to prune the large space of possible motifs. MITRA is very resource intensive and requires large amounts of memory.

A host of techniques have been developed to find sequences in a time series database that are similar to a given query sequence [33]-[36]. However, there is little published work in finding motifs in time series databases. Time series data such as stock prices, economic indexes, time varying measurements from sensors and medical signals like ECG's can be mined for motifs, and all have compelling applications. Patel et al. [18] show that time series data can be discretized and converted into a sequence over a fixed alphabet and mined using existing motif mining algorithms. Another algorithm that finds frequent trends in time series data was proposed by Udechukwu, Barker, and Alhajj in [37]. However, these algorithms mine for *exact* frequent patterns, and are difficult to employ in the case of noisy datasets. Chiu et al. describe an algorithm in [20] (based on the Random Projections algorithm [15]) which accounts for noise in the data. However, this algorithm is also limited to a simple mismatch based noise model.

3,THE MODEL

A critical aspect of the motif mining problem is defining the model under which two or more sequences are considered to match (approximately). Developing such models poses an interesting challenge: On the one hand, we want a model that is robust enough to detect the occurrence of a pattern even in the presence of noise, and on the other hand, we do not want it to be so general that it matches unrelated subsequences. Since different applications may have different criteria for how to strike this balance, a natural approach is to develop a flexible model with a few intuitive parameters that can be set by the user based on the application characteristics. In this section, we present a powerful new model for motifs that can be used for pattern mining in many different domains.

Throughout this section, we will assume that the input sequence is composed of symbols from a discrete alphabet set. However, our methods can also be applied to continuous time series datasets by converting such datasets into a *symbolic* sequence dataset by simply discretizing the numeric data. In fact such a transformation is frequently carried out for mining continuous time series datasets [18], [20].

We call our motif model the (L, M, s, k) model after the four parameters that determine it. L is the length of the motif, M is a distance matrix that is used to compute the similarity between two strings, s is the maximum distance threshold within which two strings are considered similar, and finally, k is the minimum support required for a pattern to qualify as a motif.

The (L, M, s, k) model is a very intuitive and powerful model, and permits the user a lot of flexibility in making the right tradeoff between specificity and noise tolerance of a model. As we describe below, much of this power comes from the ability to use any matrix M as the distance matrix. This property makes it useful for a variety of complex motif mining tasks. The matrix M allows us to define a *distance penalty* when a symbol X in the model matches a symbol Y in the data sequence. The penalty is specified by M(X, Y), an entry in the matrix. The total distance between the two strings is computed by summing the distance penalties of the corresponding symbols.

That is, if A = $a_1 a_2 a_3 ... a_n$ and B = $b_1 b_2 b_3 ... b_n$ are two strings, then the distance between A and B under this model is $d(A, B) = \Sigma^{n} M(a, b)$. Formally speaking, a string S is an (L, M, s, k) motif if there exist at least k strings $T_1, ..., T_k$ in the database such that each of them is of length L, and $d(S, T_i) \leq s$, where $d(A, B) = \Sigma^{n} M(a, b)$ is the distance function. Every string S that satisfies the above is an (L, M, s, k) motif. Note that the string S need not actually appear in the database for it to qualify as a motif. Only the instances T_i need to be in the database.

Protein motif mining is an example of a domain which requires a matrix based measure of similarity. Finding regions in protein sequences that appear frequently in different proteins is useful in inferring the functional sites in proteins. As in the case of DNA, the patterns in protein sequences do not repeat exactly. The instances of the pattern usually differ from the model in a few positions. To complicate things further, not all mismatches are equally bad. Some amino acids are very similar to each other, while some are very different. For instance Alanine and Valine are both hydrophobic amino acids, while Glycine and Serine are both hydrophilic. The matrix can be used to award a small penalty for M(X,Y) when X and Y are similar (Alanine and Valine, for instance) and a larger penalty otherwise (say, Alanine and Glycine) [2]. Popular substitution matrices such as PAM [1] and BLOSUM [2] can easily be used in our model.

Next, we demonstrate how this model can also be applied to the stock price example of Section I. Suppose that we had normalized the data for firm ABC. Assume that the normalized stock price values are between 0-10. If we discretized them to integers, we could use letters A - K to represent 0 - 10. Suppose further that we wanted to find sequences of length 10 that appeared (approximately) in the database at least 20 times. If we wanted to use the sum of squared differences as the distance metric to check for similarity, we can simply use a matrix where M(X,Y) is set to $(v(X) - v(Y))^2$ where v(X) is the numerical value corresponding to the symbol X. Using this matrix, we can specify that an instance matches the model if the Euclidean distance between them is within a given threshold. We model this problem as a (10, M, s, 20) motif finding

problem, where s is an appropriately chosen similarity threshold.

The matrix can be adapted to allow other kinds of models. In fact, the matrix approach lets us simulate any L_p-norm (Manhattan distance, Euclidean distance, etc.). If we wanted to match two sequences only if the corresponding values (in the two sequences) were within 2 units of each other, (the /-error tolerance model from Section I), we would just set $M(X,Y) = 0$ where $|v(X) - v(Y)| \leq 2$, and ∞ everywhere else. In general, any measure that can be computed in an incremental fashion by comparing the symbols in the corresponding positions can be simulated by constructing an appropriate distance penalty matrix.

We now discuss two special cases of the (L, M, s, k) model that are commonly used in computational biology and other domains - the (L, d, k) and (L, f, d, k) models.

A. Special Case: The (L, d, k) Model

The (L, d, k) model is a mismatch based model commonly used in computational biology for finding DNA motifs. The distance measure between two strings is the Hamming dis- tance, or merely the number of mismatches. The (L, d, k) model is parameterized by the length of the string that we want to find (L), the maximum Hamming distance (d), and the support (k). The parameter d controls the amount of noise we wish to tolerate.

The (L, d, k) model is a special case of the (L, M, s, k) model. It can easily be simulated by a matrix by setting $M(X, Y) = 1$ if $X \neq Y$ and $M(X, Y) = 0$ if $X = Y$. This way, the distance function simply counts the number of mismatches. We set s to d and use the k from (L, d, k) as our minimum support.

One of the applications of this model is in the field of computational biology. The (L, d, k) model and its derivatives have been considered a good fit for DNA regulatory motifs [32]. Briefly, the related problem of using this model to find regulatory motifs in DNA is as follows: Biologists today are interested in understanding how different genes in the genome are regulated and the way they interact with each other. To this end, biologists often study genes that exhibit similar expression patterns to extract clues about the proteins that control their expression. It is believed that genes that are co-regulated by the same protein (called a transcription factor) share some signal that allows the transcription factor to recognize the gene and turn it on. This signal is usually present in the region upstream of a gene (within a few thousand base pairs) called the promoter region. The signature is usually a short string of DNA 6-15 bases long. As is often the case in biology, these signatures are seldom identical, and differ in a few positions from one gene promoter region to another. Finding this noisy signature that is common across all the genes is a very important step towards locating the binding site for the transcription factor. Modeling the set of promoter regions as our database, and the signature binding site as an (L, d, k) pattern, we can simply apply the FLAME

algorithm to solve this problem. We show in Section VI, that this is indeed an effective approach.

In most practical situations we don't know the exact value of L, and therefore, we might have to try several values. In the case of DNA regulatory patterns, we know that the signature is usually between 6 to 15 bases long, and therefore we can try these lengths with varying number of mismatches.

The (L, d, k) model can also be used in other applications to tolerate an occasional burst of noise. If two sequences were identical except for the addition of a noise spike in one of them, they will match under a 1-mismatch model. Consider the two sequences shown in Figure 2. The two bold segments are identical except for the single spike in the lower sequence. Such spikes may occur due to measurement error or other reasons, and an (L, d, k) model will be able to tolerate this noise and correctly match the two sequences.

We show in Section VI that FLAME is faster than several existing algorithms that can only find (L, d, k) motifs.

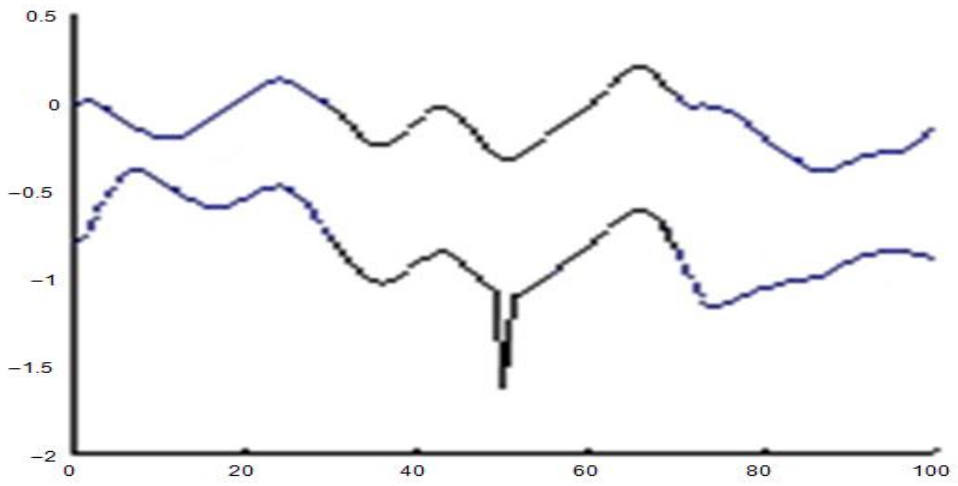

Fig. 2. Potential use of the (L, d, k) model - the lower segment is identical to the upper segment except for the single spike. The (L, d, k) model can match these.

B. Special Case: The (L, f, d, k) Model

The (L, f, d, k) builds on the (L, d, k) model to include positional constraints on the mismatches. We introduce this model using an example: Consider the three sequences {ABCD, ACCD, ABCA}. If ABCD is the model sequence, the other two sequences are within one mismatch of the motif, so these sequences would constitute a (4,1,3) motif in the (L, d, k) model. Now consider the sequences {ABCD,ACCD, ADCD}. This set also forms a (4,1,3) motif, but the mismatches,

whenever they occur, are always in position two (AcCD, AdCD). The (L, f, d, k) model allows us to specify the number of fixed – position mismatches (f) along with just the number of free mismatches (d). We look for all model strings whose instances always differ from it (if they differ at all) in the same positions.

The (L, f, d, k) model is also a special case of the (L, M, s, k) model. In order to model the fixed position mismatches, we simply augment the alphabet A with a wildcard symbol, say "?". For symbols in A, the distance matrix M is as in the (L, d, k) model, with M(X, Y) = 1 if X 6= Y and zero everywhere else. The wildcard symbol is allowed to match any symbol with no penalty, so we set M(?, X) = 0 for all X. FLAME considers all model strings of length L over the augmented alphabet such that there are at most f occurrences of the wildcard symbol. This way, the (L, M, s, k) model can simulate the (L, f, d, k) model.

We illustrate the advantage of being able to use positionally biased scoring with an example. Consider a DNA dataset consisting of 5 sequences,each of length 500. Assume that each sequence has in it the motif GTGAACAC, and each instance of the motif has a mismatch at the fifth position. In other words, the dataset contains an (8, 1, 0, 5) motif. Note that an (8, 1, 0, 5) motif is also an (8,1,5) motif in the (L, d, k) model since a free mismatch can capture a fixed mismatch. If we use the (L, d, k) model to retrieve this pattern, we will end up with many extraneous hits that might not be meaningful. When we search for an (8,1,0,5) pattern, FLAME (correctly) returns the result GTGA?CAC. On the other hand,if we search for (8,1,5), FLAME returns several additional hits that satisfy (8,1,5) but not (8,1,0,5). A post-processing step is needed to check if these are actually fixed position mismatch motifs. An (L, d, k) model can be used to simulate an (L, f, p, k) model if f + p = d with some post processing. However, as we will explain in Section V-B , using an (L, f, d, k) model produces a huge cost saving when compared to (L, d + f, k) with post- processing.

4,THE FLAME ALGORITHM

In this section, we describe the FLAME algorithm, which can be used to find (L, M, s, k) motifs. For ease of exposition, we explain the algorithm using an (L, d, k) model, and then describe how we extend it to the full-fledged (L, M, s, k) model.

Recall that an (L, d, k) motif is a string of length L that occurs k times in the dataset, with each occurrence being within a Hamming distance of d from the model string. Given, L , d, and k, a naive algorithm is to consider all possible strings of length L over the alphabet (the space of all models), and compute the support for each of them by scanning the dataset. This algorithm is exponential and becomes infeasible with large L and d values. One might be tempted to improve this method by considering only those strings of length L that actually occur in the dataset. However, this approach

might miss motifs as the model string might not actually occur in the dataset even once. Assume that we are looking for a (6, 2, 3) pattern, and that the instances of this pattern in the dataset are FFCDEF, ABFFEF, and ABCDAA. Each instance is at a distance of 2 from the model ABCDEF, but the distance between any two instances is 4. If we consider only instances from the dataset (which need not contain ABCDEF), then we will not find the motif.

The approach we take in FLAME explores the space of *all* possible models. In order to carry out this exploration in an efficient way, we first construct two suffix trees: a count suffix tree on the actual dataset (called the *data suffix tree*), and a suffix tree on the set of all possible model strings (called the *model suffix tree*). This second set is typically the set of all strings of length L over the alphabet. As we describe below, the model suffix tree helps guide the exploration of the model space in a way that avoids redundant work. The data suffix tree helps us quickly compute the support

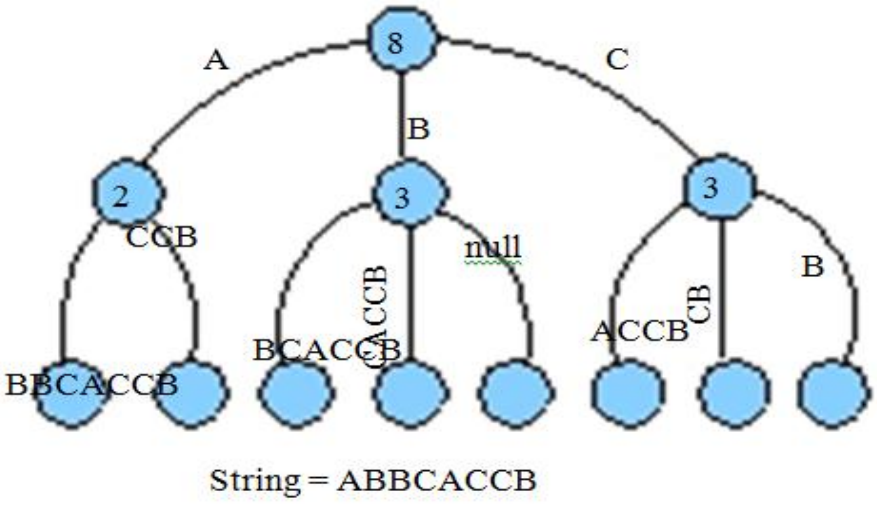

Fig 3: A count suffix tree on the string, ABBCACCB. The count are indicated inside the node. of a model string. Recall that a count suffix tree is merely a suffix tree in which every node contains the number of leaves in the subtree rooted at that node. Every node contains the number of occurrences of the string corresponding to that node. Essentially, the data suffix tree combines the work common to finding the support for models like ABCDE and ABCDF (having a common prefix) and perform it only once.

Since the second suffix tree (built on all possible model strings) can be extremely large, FLAME does not actually construct this suffix tree. Rather, it algorithmically generates portions of this tree as and when needed. FLAME then explores the model space by traversing this (conceptual) model suffix tree. Using the suffix tree on the dataset, FLAME computes support at various nodes in the model space and prunes away large

portions of the model space that are guaranteed not to produce any results under the model. This careful pruning (described in more detail below), ensures that FLAME does not waste any time exploring models that do not have enough support. The FLAME algorithm simply stops when it has finished traversing the model suffix tree and outputs the model strings that had sufficient support.

To understand our strategy of pruning the model suffix tree, consider the following example: Assume that the dataset consists of sequences over the alphabet {A,B,C,D,E}. The dataset and the values of L, d, and k are specified as input. All the strings of length L starting with the symbol A form a subset of the model space. We call this the A partition. This partition corresponds to all the nodes in the model suffix tree under the subtree corresponding to node A. This partition is further divided into sub-partitions with prefix AA, AB, AC, AD, and AE. These partitions continue on for L levels, and at the last level, we have only one model string for each partition.

5, EVALUATION

In this section, we present results from various experiments that were designed to test the effectiveness and performance of FLAME. We also compare FLAME with pattern mining algorithms from different application domains. Most existing algorithms can only work with (L, d, k) motifs and do not support the more general (L, M, s, k) model. Therefore, we carry out the comparison between FLAME and these existing methods using only the (L, d, k) model. Since we do not have a competing algorithm to compare the performance of FLAME on (L, M, s, k), we present a detailed analysis of the performance of FLAME as different parameters in (L, M, s, k) are varied.

We use a variety of datasets for our experiments:

Snake: This is a snake protein dataset from [38] that was considered for subsequence mining in [6]. It consists of 352 different snake venom protein sequences of varying lengths. The size of the dataset is about 28,000 symbols. The alphabet of amino acids (that make up the proteins) is of size 20. Such protein datasets are often analyzed in bioinformatics to find common patterns that might provide insights into their function.

IBM: This dataset contains second by second average price of IBM stock for all the trading days in December 1999 [39]. To reduce the noise in the detailed dataset, we preprocess the data using the following standard data processing techniques that are designed to deal with short term volatility in stock price information [40]: First, the data is converted into a minute wise average price using a sliding window. And next, the price values are transformed into a percentage change with respect to the price in the previous minute. This technique is routinely used to compare movement data across different stocks that have a different face value. The resulting dataset contained 21 sequences from 21 days, each

of length approximately 400 numbers, totaling 8,400 numbers.

Synthetic: In order to fully explore the space of data sizes and alphabet sizes, we use a synthetic data generation method that has been extensively used in several previous efforts [13]- [15], [41]. The data is generated as follows: Given the alphabet size, the number of sequences, and the size of each sequences, we generate random sequences by uniformly drawing symbols from the alphabet. We then randomly choose k sequences and *implant* a pattern of length L with d mismatches at random positions in each of the k sequences. This results in a dataset containing an (L, d, k) motif. The sizes of datasets we generate are comparable to those used in previous related papers [13]- [15], [41].
All the experiments in this section were performed on a 2.8 GHz Intel Pentium 4 processor with 2 GB of main memory. The operating system was Fedora Core 4 Linux, kernel version 2.6.11. The YMF implementation was obtained from [42], Weeder from [43], and Random Projections (RP) from [44]. RP is a widely used technique for motif mining (and has been applied in various domains like time-series mining and DNA motif mining), and YMF and Weeder are the leading popular DNA motif mining methods [32].

As discussed in Section I, FLAME solves a different problem compared to traditional sequence mining methods like cSPADE [45] and CloSpan [7]. Nevertheless, for completion, we modified CloSpan to mine for contiguous subsequences (which improved its performance by 3 orders of magnitude) and adapted cSPADE to mine contiguous motifs. The comparison is presented in the extended version of this paper [11]. The results show that FLAME outperforms these methods by an order of magnitude or more and scales significantly better.

VII, CONCLUSIONS

In this paper, we presented a powerful new model: (L, M, s, k) for motif mining in sequence databases. The (L, M, s, k) model subsumes several existing models and provides additional flexibility that makes it applicable in a wider variety of data mining applications. We also presented FLAME, a flexible and accurate algorithm that can find (L, M, s, k) motifs. Through a series of experiments on real and synthetic datasets, we demonstrate that FLAME is a versatile algorithm that can be used in several real motif mining tasks. We also show that FLAME outperforms existing time series mining algorithms (Random Projections) by more than an order of magnitude. FLAME is also superior to motif finding algorithms used in computational biology (more accurate than Weeder, significantly faster than YMF). We also presented experiments which show that FLAME can scale to handle motif mining tasks that are much larger than attempted before.

REFERENCES

[1] M. O. Dayhoff, R. M. Schwartz, and B. Orcutt, "A Model for Evolutionary Changes in Proteins," *Atlas of Protein Sequence and Structure*, vol. 5, pp. 345-352, 1978.

[2] S. Henikoff and J. Henikoff, "Amino Acid Substitution Matrices from Protein Blocks," *National Academy of Sciences, USA*, vol. 89, no. 22, pp. 10915-9, 1992.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *VLDB*, 1994, pp. 487-499.

[4] ——, "Mining Sequential Patterns," in *ICDE*, 1995, pp. 3-14.

[5] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning*, vol. 42, no. 1/2, pp. 31-60, 2001.

[6] J. Wang and J. Han, "BIDE: Efficient Mining of Frequent Closed Sequences," in *ICDE*, 2004, pp. 79-90.

[7] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining Closed Sequential Patterns in Large Datasets," in *SDM*, 2003.

[8] J. Yang, W. Wang, P. S. Yu, and J. Han, "Mining Long Sequential Patterns in a Noisy Environment," in *SIGMOD*, 2002, pp. 406-417.

[9] M. J. Zaki, "Sequence Mining in Categorical Domains: Incorporating Constrains," in *CIKM*, 2000, pp. 442-429.

[10] J. Pei, J. Han, and W. Wang, "Mining Sequential Patterns With Constraints in Large Databases," in *CIKM*, 2002, pp. 18-25.

[11] A. Floratou, S. Tata, and J. M. Patel, "Finding Hidden Patterns in Sequences, Tech. Rep. http://www.pages.cs.wisc.edu/~floratou, 2009.

[12] S. Sinha and M. Tompa, "YMF: A Program for Discovery of Novel Transcription Factor Binding Sites by Statistical Overrepresentation," *Nucleic Acids Research*, vol. 31, no. 13, 2003.

[13] G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole, "Weeder Web: Discovery of Transcription Factor Binding Sites in a Set of Sequences From Co-Regulated Genes," *Nucleic Acids Research*, vol. 32(Web Server issue), pp. W199-W203, 2004.

[14] E. Eskin and P. A. Pevzner, "Finding Composite Regulatory Patterns in DNA Sequences," in *ISMB*, 2002, pp. S354-63.

[15] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," *Journal Computational Biology*, vol. 9, no. 2, pp. 225-242, 2002.

[16] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule Discovery From Time Series," in *KDD*, 1998, pp. 16-22.

[17] S. Hoppner, "Discovery of Temporal Patterns - Learning Rules about the Qualitative Behaviour of Time Series," in *5th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2001, pp. 192-203.

[18] P. Patel, E. Keogh, J. Lin, and S. Lonardi, "Mining Motifs in Massive Time Series Databases," in *ICDM*, 2002, pp. 370-377.

[19] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. Kaeli, "Subsequence Matching on Structured Time Series Data," in *SIGMOD*, 2005, pp. 682-693.

[20] B. Y.-C. Chiu, E. J. Keogh, and S. Lonardi, "Probabilistic Discovery of Time Series Motifs," in *KDD*, 2003, pp. 493-498.

[21] W. Wang and J. Yang, *Mining Sequential Patterns from Large Data Sets*. Springer-Verlag, 2005, vol. 28.

[22] M. Das and H. K. Dai, "A Survey of DNA Motif Finding Algorithms," *BMC Bioinformatics*, vol. 8, 2007.

[23] G. K. Sandve and F. Drabløs, "A Survey of Motif Discovery Methods in an Integrated Framework," *Biology Direct*, vol. 1, 2006.

[24] Y. Zhang and M. J. Zaki, "ExMOTIF: Efficient Structured Motif Extraction," *Algorithms for Molecular Biology*, vol. 1, no. 21, November 2006.

[25] A. M. Carvalho *et al.*, "An Efficient Algorithm for the Identification of Structured Motifs in DNA Promoter Sequences," *IEEE/ACM Transactions on computational biology and bioinformatics*, vol. 3, 2006.

[26] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, "PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth," in *ICDE*, 2001, pp. 215-224.

[27] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert, "Approaches to the Automatic Discovery of Patterns in Biosequences," *Journal of Computational Biology*, vol. 5, pp. 279-305, 1998.

[28] L. Marsan and M.-F. Sagot, "Algorithms for Extracting Structured Motifs Using a Suffix Tree with Application to Promoter and Regulatory Site Consensus Identification," *Journal of Computational Biology*, vol. 7, no. 3/4, pp. 345-360, 2000.

[29] F. Zhu, X. Yan, J. Han, and P. S. Yu, "Efficient discovery of frequent approximate sequential patterns," in *ICDM*, 2007.

[30] T. L. Bailey and C. Elkan, "Unsupervised Learning of Multiple Motifs in Biopolymers using EM," *Machine Learning*, vol. 21, no. 1-2, pp. 51-80, 1995.

[31] W. Thompson, E. C. Rouchka, and C. E. Lawrence, "Gibbs Recursive Sampler: Finding Transcription Factor Binding Sites," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3580-3585, 2003.

[32] M. Tompa *et al.*, "Assessing Computational Tools for the Discovery of Transcription Factor Binding Sites," *Nature Biotechnology*, vol. 23, pp. 137-144, 2005.

[33] A. W.-C. Fu, E. J. Keogh, L. Y. H. Lau, and C. A. Ratanamahatana, "Scaling and Time Warping in Time Series Querying," in *VLDB*, 2005, pp. 649-660.

[34] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering Similar Multidimensional Trajectories," in *ICDE*, 2002, pp. 673-684.

[35] L. Chen, M. Tamer Ozsu, and V. Oria, "Robust and Fast Similarity Search for Moving Object Trajectories," in *SIGMOD*, 2005, pp. 491- 502.

[36] Y. Zhu and D. Shasha, "Warping Indexes with Envelope Transforms for Query by Humming," in *SIGMOD*, 2003, pp. 181-192.

[37] A. Udechukwu, K. Barker, and R. Alhajj, "Discovering all frequent trends in time series," in *Proc. of Winter Int. Sym. on Information and Comm. Tech.*, vol. 58, 2004, pp. 1-6.

[39]"Data Sets from Analysis of Financial Time http://www.gsb.uchicago.edu/fac/ruey.tsay/teaching/fts/.

[40] R. S. Tsay, *Analysis of Financial Time Series*, 1st ed.

[41] P. A. Pevzner and S.-H. Sze, "Combinatorial Approaches to Finding Subtle Signals in DNA Sequences," in *ISMB*, 2000, pp. 269-278.

[42] "YMF Source Code," http://bio.cs.washington.edu/software.html.
[43] "Weeder Source Code," http://www.pesolelab.it/Tool/ind.php.
[44] "Random Projections Source Code," http://www.cse.wustl.edu/ jbuhler/pgt/.
[45] "cSPADE Source Code," http://www.cs.rpi.edu/ zaki/software/.
[46] S. Tata, R. A. Hankins, and J. M. Patel, "Practical Suffix Tree Construction," in *VLDB*, 2004, pp. 36-47.
[47] A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," in *VLDB*, 1999, pp. 518-529.
[48] T. L. Bailey and C. Elkan, "Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Biopolymers," in *ISMB*, 1994, pp. 28-36.
[49] "TRANSFAC," http://www.gene-regulation.com/pub/databases.html.