Efficient Discovery of Frequent Approximate Sequential Patterns

Feida Zhu[†] Xifeng Yan[‡] Jiawei Han[†] Philip S. Yu[‡]

[†]University of Illinois at Urbana-Champaign
{feidazhu, hanj}@uiuc.edu

[‡]IBM T. J. Watson Research Center
xifengyan, psyu@us.ibm.com

Abstract

We propose an efficient algorithm for mining frequent approximate sequential patterns under the Hamming distance model. Our algorithm gains its efficiency by adopting a "break-down-and-build-up" methodology. The "break-down" is based on the observation that all occurrences of a frequent pattern can be classified into groups, which we call strands. *We developed efficient algorithms to quickly mine out all strands by iterative growth. In the "build-up" stage, these strands are grouped up to form the support sets from which all approximate patterns would be identified. A salient feature of our algorithm is its ability to grow the frequent patterns by iteratively assembling building blocks of significant sizes in a local search fashion. By avoiding incremental growth and global search, we achieve greater efficiency without losing the completeness of the mining result. Our experimental studies demonstrate that our algorithm is efficient in mining globally repeating approximate sequential patterns that would have been missed by existing methods.*

1 Introduction

Frequent sequential pattern mining remains one of the most important data mining tasks since its introduction in [1]. With the ubiquity of sequential data, it has found broad applications in customer analysis, query log analysis, financial stream data analysis and pattern discovery in genomic DNA sequences in bioinformatics. Extensive research on the topic has brought about general sequential pattern mining algorithms like [11, 15, 7, 18, 13, 2] and constraint-based ones like [4, 14]. Periodic pattern mining in temporal data sequences has also been studied [6, 3].

However, all these mining algorithms follow the exact-matching sequential pattern definition. It has been shown that the capacity to accommodate approximation in the mining process has become critical due to inherent noise and imprecision in data, *e.g.*, gene mutations in genomic DNA sequence mining. The notion of approximate sequential pattern has been proposed in [8], in which an algorithm called ApproxMap is designed to mine consensus patterns. While mining consensus patterns provides one way to produce compact mining result under general distance measures, it remains a challenge how to efficiently mine the complete set of approximate sequential patterns under some distance measure which is stricter yet equally useful in many cases. The Hamming distance model, which counts only mismatches, is one of such.

We look at bioinformatics for example. The identification of repeats serves as a critical step in many biological applications on a higher level such as a preprocessing step for genome alignment, whole genome assembly and a post-processing step for BLAST queries. For repeat families that are relatively new in the evolution, the set of repeats found under the Hamming distance model captures almost the complete set. (2) The limited knowledge that biologists currently have of these repeats makes it often hard for them to evaluate the relative significance among different repeats. It is therefore worth the effort to mine the complete set. Existing tools like RepeatMasker [12] only solve the problem of pattern matching, rather than pattern discovery without prior knowledge. (3) Many research works for the repeating patterns have been on an important subtype: the tandem repeats [10], where repeating copies occur together in the sequence. However, as shown by our experiments, these methods would miss those patterns whose supporting occurrences appear globally in the entire data sequence, which account for the majority of the complete set of frequent patterns.

REPuter [9] is the closest effort towards mining frequent approximate sequential patterns under the Hamming distance model. Unfortunately, REPuter achieves its efficiency by strictly relying on the suffix tree for constant-time longest common prefix computation in seed extension. Consequently, the type of approximate patterns that REPuter is able to mine is inevitably limited. In particular, it can only discover patterns with two occurrences and mismatches at identical positions across the support set. The mining problems targeted by REPuter and us are essentially two different ones.

To uncover more interesting approximate patterns in DNA sequences, we establish a more general model for approximate sequential pattern mining problem. Our general philosophy is a "break-down-and-build-up" one based on the following observation. Although for an approximate pattern, the sequences in its support set may have different patterns of substitutions, they can in fact be classified into groups, which we call *strands*. Each strand is a set of sequences sharing a unified pattern representation together with its support. The idea is that by "breaking down" the support sets of the approximate patterns into strands, we are able to design efficient algorithms to compute them. Using a suffix-tree-based algorithm, we can in linear time mine out the initial strands, which are all the exactly matching repeats. These initial strands will then be iteratively assembled into longer strands in a local search fashion, until no longer ones can be found. In the second "build-up" stage, different strands are then grouped based on their constituting sequences to form a support set so that the frequent approximate patterns would be identified. By avoiding incremental growth and global search, we are able to achieve great efficiency without losing the completeness of the mining result. Instead of mining only the patterns repeating within a sliding window of fixed sizes, our algorithm is able to mine all globally repeating approximate patterns.

2 Problem Formulation

In our problem setting, we focus on mining approximate sequential patterns under the Hamming distance model. Hamming distance, which is defined for two strings of equal length, is the number of substitutions required to change one into the other.

Definition 1 (Hamming Distance) *For two strings $S = \langle s_1, s_2, \ldots, s_n \rangle$ and $P = \langle p_1, p_2, \ldots, p_n \rangle$ of a same length n, the Hamming distance between them is defined as*

$$Dist(S, P) = |I|, I = \{i | s_i \neq p_i, 1 \leq i \leq n\}$$

The Hamming distance between two strings S and P is denoted as $Dist(S, P)$. In our model, two sequential patterns are considered approximately the same if and only if they are of equal length and their distance is within a user-specified error tolerance. We therefore use string or substring to refer to all sequential patterns in the rest of the paper. Given a string $S = \langle s_1, s_2, \ldots, s_n \rangle$ of length n, another string $Z = \langle z_1 \ldots z_m \rangle$ is a *substring* of S if there exists an index i of S such that $z_j = s_{i+j}$ for all $1 \leq j \leq m$. In this case, S is a *superstring* of Z. We use $|S|$ to denote the length of a string S.

Given an input string S, we are interested in finding all frequent approximate substrings of S, *i.e.*, for each such substring, the set of substrings that are considered approximately the same must be sufficiently large.

Definition 2 (Frequent Approximate Substring) *Given a string S, a substring P of S is a frequent approximate substring if and only if there exists a set U of substrings of S and for each $W \in U$, $Dist(P, W) \leq |P|\delta$, and $|U| \geq \theta$, where θ is the minimum frequency threshold and δ is the error tolerance threshold. U is called the support set of P, denoted as P_{sup}.*

Notice that U is represented as a set of indices of S as all substrings in U share the same length as P.

As in frequent itemset mining, the definition of frequent approximate substring also gives rise to redundancy in the mining result. Consider the three substrings in Figure 1.

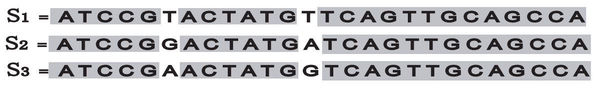
S1 = ATCCGTACTATGTTCAGTTGCAGCCA
S2 = ATCCGGACTATGATCAGTTGCAGCCA
S3 = ATCCGAACTATGGTCAGTTGCAGCCA

Figure 1.

Suppose S_1 is a frequent approximate substring, with its distances to S_2 and S_3 being both 2 in this case. If we delete the last character $'A'$ from all three strings, the resulting substring S_1 is still a frequent approximate substring with its distances to S_2 and S_3 unchanged. This remains true as we delete more characters so long as the error tolerance requirement is satisfied. It would be considered redundancy in many cases if all such shorter substrings of S_1 are also reported. Ideally, we would like to report a substring only when it can not be extended without changing its distance to some substring in its support set. In the example of Figure 1, we would like to report six frequent approximate substrings as shown in Figure 2. We therefore define the *closeness* for a frequent approximate substring.

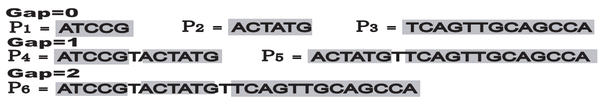

Figure 2.

Definition 3 (Closed Frequent Approximate Substring) *Given a string S, a frequent approximate substring P of S is closed if and only if there exists no frequent approximate substring Z of S such that (1) Z is a superstring of P, (2) there exists a bijection between Z_{sup} and P_{sup} such that for each $S_i \in P_{sup}$, there exists a $S'_i \in Z_{sup}$ such that S'_i is a superstring of S_i, and (3) $Dist(Z, S'_i) = Dist(P, S_i)$ for some $S_i \in P_{sup}$.*

In this paper, we study the problem of mining all closed frequent approximate substrings from a given data string. For brevity, all frequent approximate substrings mined by our algorithm are closed for the rest of the paper. A frequent approximate substring will be abbreviated as a FAS.

Formally, the frequent approximate substring mining problem (FASM) is defined as follows.

Definition 4 (FASM) *Given a string S, a minimum frequency threshold θ and an error tolerance threshold δ, the FASM problem is to find all closed frequent approximate substring P of S.*

3 Algorithm Design

In general, for a FAS P, consider any two substrings W_1 and W_2 in P_{sup}. Aligning W_1 and W_2, we observe an alternating sequence of maximal matching substrings and gaps of mismatches $Pattern(W_1, W_2) = \langle M_1, g_1, M_2, g_2, \ldots, M_k \rangle$, where $M_i, 1 \leq i \leq k$ denote the maximal matching substrings shared by W_1 and W_2. $g_i, 1 \leq i < k$ denote the number of mismatches in the i-th gap. Consider four substrings S_1, S_2, S_3 and S_4 as shown in Figure 3.

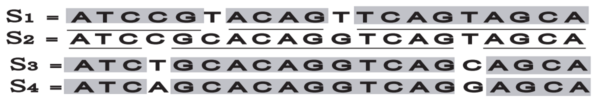
S1 = ATCCGTACAGTTCAGTAGCA
S2 = ATCCGCACAGGTCAGTAGCA
S3 = ATCTGCACAGGTCAGCAGCA
S4 = ATCAGCACAGGTCAGGAGCA

Figure 3.

In this case, $Pattern(S_1, S_2) = \langle ATCCG, 1, ACAG, 1, TCAGTTGCA \rangle$. All four substrings are of length 20. If the error tolerance threshold $\delta = 0.1$ and minimum frequency threshold $\theta = 4$, then S_2 is a FAS since the other three substrings are within Hamming distance 2 from S_2. For each substring, the bounding boxes indicate the parts that match exactly with S_2. We can therefore define the notion of a *strand*, which is a set of substrings that share one same matching pattern.

Definition 5 *A set U of substrings $U = \{S_1, \ldots, S_k\}$ is a* strand *if and only if for any two pairs of substrings $\{S_{i_1}, S_{j_1}\}$ and $\{S_{i_2}, S_{j_2}\}$ of U, $Pattern(S_{i_1}, S_{j_1}) = Pattern(S_{i_2}, S_{j_2})$.*

By definition, all the substrings in a strand U share the same alternating sequence of maximal matching substrings and gaps of mismatches, which we call $Pat(U)$. We use $|Pat(U)|$ to denote the length of the substrings in U. We use $Gap(U)$ to denote the number of gaps in $Pat(U)$ and $Miss(U)$ to denote the number of total mismatches in $Pat(U)$. Given a strand U with its corresponding matching pattern $Pat(U) = \langle M_1, g_1, \ldots, M_k \rangle$, $Dist(S_i, S_j) = Miss(U) = \sum_{i=1}^{k-1} g_i$, for all $S_i, S_j \in U$ and $i \neq j$. All substrings in a strand share a same distance from one another. Define $Plist(U)$ to be the *support set* of a strand U, *i.e.*, $Plist(U)$ is the set of indices where each substring in U occurs. A strand U is represented by the pair $\langle Pat(U), Plist(U) \rangle$.

We call a strand U *valid* if the distance between any two substrings of U satisfy the user-specified error tolerance threshold, *i.e.*, $Miss(U) \leq |Pat(U)|\delta$. Similar to the notion of the closeness of a FAS, we have the definition for the closeness of a strand. A strand U is *closed* if and only if there exists no strand U' such that (1) there exists a bijection between the set of substrings of U and U' such that for each $P \in U$, there is a $P' \in U'$ and P' is a superstring of P, and (2) $Miss(U) = Miss(U')$.

A FAS could belong to multiple strands. For any given FAS, the observation is that its support set is exactly the union of all its closed valid strands.

We therefore have the following approach to decide if a given substring P is a FAS: Find all the closed valid strands of P and let the union of them be X. P is a FAS if and only if the cardinality of X is at least θ. Consider the example in Figure 3 in which the error tolerance is 0.1 and minimum frequency threshold is 4. Both strands $\{S_1, S_2\}$ and $\{S_2, S_3, S_4\}$ are valid. Suppose these two strands are also closed, then combining them we get a support set of size 4, satisfying the frequency requirement. As such, S_2 is a FAS.

Our algorithm solves the FASM problem in two steps.

1. **Growing Strand**
   Compute a set of closed valid strands initially. The set of initial strands is the set of all maximal exact repeats. More precisely, for each initial strand U, $Pat(U) = \langle M_1 \rangle$, $Miss(U) = 0$ and U is closed. These initial strands are computed by $InitStrand$ using the suffix tree of the input sequence S. Similar approach has been used in REPuter [9] to mine exact repeats. By a linear-time suffix tree implementation as in [5], we are able to identify all initial strands in time linear to the input size. To mine out all closed valid strands, we iteratively call the following procedure to grow the current set of strands until no new strands are found: We scan the entire tape and, for each strand encountered, checks on both ends to see if the current strand can be grown by assembling neighboring strands. Let the result set be X.

2. **Grouping Strand**
   Once we have mined out all the closed valid strands in the first step, we compute the support set for each frequent approximate substring. The idea of grouping the strands is the following. Given the set X of all closed valid strands, we construct a substring relation graph G from X. The vertex set is all the substrings in the strands of X, each vertex representing a distinct substring. There is an edge between two substrings if and only if the Hamming distance between two substrings is within the error tolerance. Since all the substrings in one valid strand share the same distance among each

other and the distance is within the error tolerance, all corresponding vertices in G form a clique. After scanning all the strands in X, we would construct a graph G which is a union of cliques. Then by our observation, a substring is a frequent approximate substring if and only if the degree of the corresponding vertex is greater than or equal to the minimum frequency threshold.

We are able to prove that our algorithm would generate the complete set of FASs. The sketch of the proof is as follows. We first prove that we would generate all closed valid strands by induction on the number of gaps of the strands. Our observation that the support set of a FAS P is the union of all sets of P's closed valid strands then tells us that we could compute the support sets of all FASs and identify them by our strand grouping algorithm. The formal proofs are omitted due to space limit.

3.1 Local Search

One salient feature of our algorithm is that only local search is performed when checking on both ends of a strand when growing strands. We therefore need to determine the distance to check, which we denote as d. If d is set to be too big, then in the worst case, we would scan the entire data string each time we check for a strand. The running time would then be $\Omega(|X|^2)$, where X is the set of all valid strands. On the other hand, if d is set to be too small, we could fail to guarantee the completeness of the mining result. Consider the following example in figure 4. Suppose we have two valid strands U_1 and U_2 such that $|Pat(U_1)| = 20, Miss(U_1) = 0$ and $|Pat(U_2)| = 40, Miss(U_2) = 0$. There is a gap of 7 mismatches between them. Suppose the error tolerance is $\delta = 0.1$. Notice that a valid strand U can accommodate further mismatches on either ends up to a distance of $|Pat(U)|\delta - Miss(U)$. Then U_1 can accommodate $d_1 = 2$ extra mismatches and U_2 can accommodate $d_2 = 4$ extra mismatches. However, as Figure 4 shows, the tricky part is that if we only search forward d_1 from U_1 and backward d_2 from U_2, we would fail to identify the chance to assemble them due to the fact that the gap is larger than the sum of d_1 and d_2. Even if we search forward from U_1 for a distance that doubles d_1, we could still miss U_2. Fortunately, searching backward from U_2 for a distance of $2d_2$ would let us reach U_1.

Then how to decide on the value of d such that we would guarantee the completeness of the mining result, and at the same time, scan as small a portion of the data string as possible? It turns out we have the following theorem to help determine the value for d.

Theorem 3.1 *Given the error tolerance δ and a strand U, searching for a distance $d = 2(|Pat(U)|\delta - Miss(U))/(1-\delta)$ would guarantee the completeness of the mining result.*

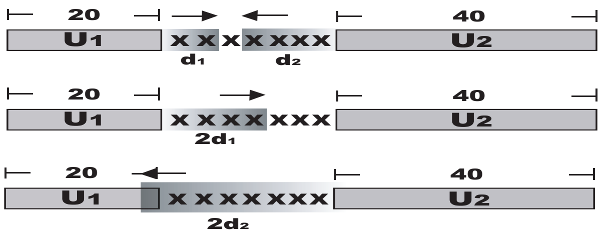

Figure 4. Assembling two strands U_1, U_2

Theorem 3.1 tells us that we don't have to search too far to guarantee the completeness. In fact, it is easy to observe that we at most search twice the distance of an optimal algorithm. Notice that any strand encountered within a distance of $\hat{d} = (|Pat(U)|\delta - Miss(U))/(1-\delta)$ can be assembled with the current strand to form a new valid strand, since the current strand itself can accommodate all the mismatches in a gap of length $\hat{d}$. As such to guarantee a complete mining result, any algorithm would have to check at least a distance of $\hat{d}$. We therefore check at most twice the distance of an optimal algorithm.

4 Performance Study

We used a real soybean genomic DNA sequence, CloughBAC, for our experiment. CloughBAC is 103334 base pairs in length. When the error tolerance δ is set as 0.1 and the minimum frequency threshold θ is set as 3, there are altogether 182046 closed approximate sequences of length at least 5. The longest closed approximate sequence is of length 995. Figure 5 shows the set of closed approximate

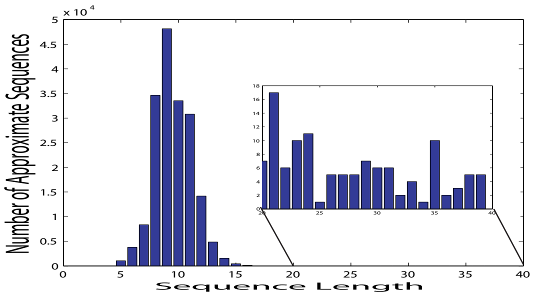

Figure 5. Sequences of size up to 40 bps

sequences of size up to 40 while Figure 6 shows the rest of the mining result, which are of size from 40 to 995. It can be observed that, in this particular soybean genomic DNA sequence, the approximate sequences are dense around the size of 10 and become sparse from size 15 to form a long tail.

We define the *spread* for an approximate sequence to be the distance between the index of its first occurrence and

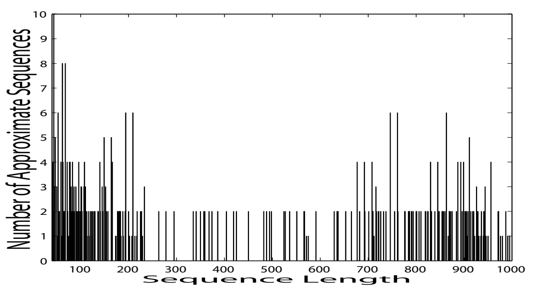

Figure 6. Sequences of size from 40 to 1000 bps

that of its last occurrence. A globally repeating approximate sequence has a large spread since its occurrences are not confined to a particular portion of the data sequence. As such, the larger the spread, the harder it is to discover the sequence by a sliding-window-based method. The spreads of all the approximate sequences in the mining result are plotted in Figure 7. It is evident that the majority of them actually have spreads comparable to the length of the orginal data sequence. The advantage of our mining approach compared against a sliding-window-based one manifests itself in the fact that even a sliding window half the size of the original data sequence would discover all the occurrences of only 30% of the complete mining result.

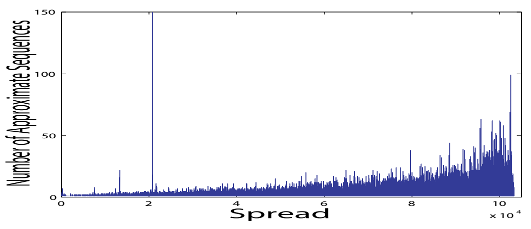

Figure 7. The spread of the mining result

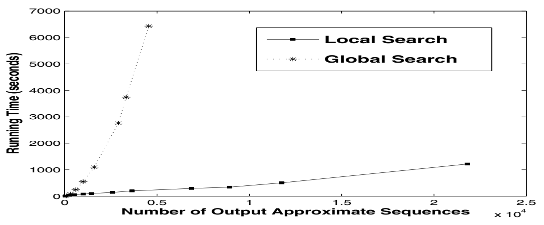

Figure 8. Run time

Figure 8 shows the running time of our algorithm as the number of output approximate sequence increases. It is compared against the one without the local search technique to demonstrate its importance in boosting the mining efficiency. The running time of our algorithm is observed to be linear in the output size. Figure 9 illustrates the runtime performance with varied error tolerance δ. The bar chart, with its y-axis on the right side of the figure, shows the corresponding numbers of output sequences as δ increases. More lenient error tolerance results in more output sequences and consequently a longer running time. Figure 10 illustrates the runtime performance with varied minimum frequency threshold θ. The bar chart, with its y-axis on the right side of the figure, shows the corresponding numbers of output sequences as θ increases. Observe that as the minimum frequency threshold increases, the output size decreases sharply while the running time almost remains the same. This is because regardless of the minimum frequency threshold for the output, all sequences with at least two occurrences have to be computed during the strand growing stage, which is responsible for most of the mining cost. It is only in the strand grouping stage that a greater minimum frequency threshold helps to reduce the running time. The influence of θ on the mining cost is therefore less significant.

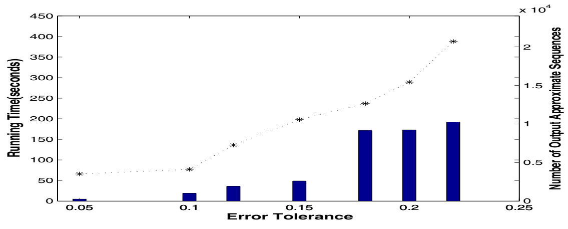

Figure 9. Run time with varied δ

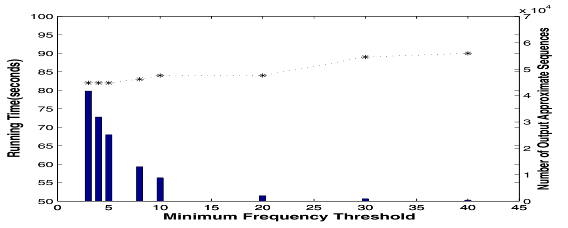

Figure 10. Runtime with varied θ

5 Related Work

A succession of sequential pattern mining algorithms have been proposed since [1], including [15], SPADE [18], PrefixSpan [13] and SPAM [2]. There are also constraint-based ones like [4, 14]. CloSpan [17] follows the candidate maintenance-and-test approach and uses techniques like *CommonPrefix* and *Backward Sub-Pattern Pruning*. BIDE [16] improves scalability by avoiding candidate maintenance and applying *BI-Directional Extension*. When approximation is taken into consideration in the frequent sequential pattern definition, the size of the mining result could be prohibitively huge under a general distance measure. ApproxMap [8] approached this problem by mining instead the consensus patterns, which are a subset of long and representative patterns. Algorithms in the bioinformatics community have been focusing on approximate pattern matching and generate popular tools like RepeatMasker [12]. Most of these algorithms target at finding tandem repeats. REPuter [9] uses suffix tree to find maximal exact repeats and employs a suffix-tree-based constant time longest common prefix algorithm to extend them. However, REPuter cannot discover patterns with more than two occurrences and mismatches present at different positions across the support.

6 Conclusions

We propose the definition of *closed* frequent approximate sequential patterns to solve the problem of mining the complete set of frequent approximate sequential pattern mining under the Hamming distance. Our algorithm is based on the notion of classifying a pattern's support set into *strands*. We combine a suffix-tree-based initial strand mining and iterative strand growth. We adopt a local search optimization technique to reduce time complexity which at the same time guarantees the completeness of the mining result. Our performance study shows that our algorithm is able to mine out globally repeating approximate patterns in biological genomic DNA data with great efficiency.

Acknowledgement

We are grateful to Prof. Matt Hudson and Kranthi Varala for providing us with the genome sequence data and many helpful discussions and suggestions.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In ICDE'95, pages 3–14.

[2] J. Ayres, J. Flannick, J. E. Gehrke, and T. Yiu. Sequential pattern mining using bitmap representation. In KDD'02, pages 429–435.

[3] C. Bettini, X. S. Wang, and S. Jajodia. Mining temporal relationships with multiple granularities in time sequences. *Bull. Technical Committee on Data Engineering*, 21:32–38, 1998.

[4] M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In VLDB'99, pages 223–234.

[5] D. Gusfield. *Algorithms on Strings, Trees and Sequences, Computer Science and Computation Biology*. Cambridge University Press, 1997.

[6] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In ICDE'99, pages 106–115.

[7] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: Frequent pattern-projected sequential pattern mining. In KDD'00, pages 355–359.

[8] H.-C. Kum, J. Pei, W. Wang, and D. Duncan. ApproxMap: Approximate mining of consensus sequential patterns. In SDM'03.

[9] S. Kurtz, J. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. Reputer: The manifold applications of repeat analysis on a genomic scale. In *Nucleic Acids Research*, number 22, pages 4633–4642, 2001.

[10] G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, number 684, pages 120–133,1993.

[11] F. Masseglia, F. Cathala, and P. Poncelet. The PSP approach for mining sequential patterns. In PKDD'98, pages 176–184.

[12] Institute for Systems Biology. Repeatmasker. In http://www.repeatmasker.org/webrepeatmasker-help.html, 2003.

[13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In ICDE'01, pages 215–224.

[14] J. Pei, J. Han, and W. Wang. Constraint-based sequential pattern mining in large databases. In CIKM'02, pages 18–25.

[15] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In EDBT'96, pages 3–17.

[16] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In ICDE'04, pages 79–90.

[17] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In SDM'03, pages 166–177.

[18] M. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 40:31–60, 2001.