MACFP: Maximal Approximate Consecutive Frequent Pattern Mining under Edit Distance

Jingbo Shang Jian Peng Jiawei Han
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA.
{shang7, jianpeng, hanj}@illinois.edu

Abstract

Consecutive pattern mining aiming at finding sequential patterns substrings, is a special case of frequent pattern mining and has been played a crucial role in many real world applications, especially in biological sequence analysis, time series analysis, and network log mining. Approximations, including insertions, deletions, and substitutions, between strings are widely used in biological sequence comparisons. However, most existing string pattern mining methods only consider hamming distance without insertions/deletions (indels). Little attention has been paid to the general approximate consecutive frequent pattern mining under edit distance, potentially due to the high computational complexity, particularly on DNA sequences with billions of base pairs. In this paper, we introduce an efficient solution to this problem. We first formulate the Maximal Approximate Consecutive Frequent Pattern Mining (MACFP) problem that identifies substring patterns under edit distance in a long query sequence. Then, we propose a novel algorithm with linear time complexity to check whether the support of a substring pattern is above a predefined threshold in the query sequence, thus greatly reducing the computational complexity of MACFP. With this fast decision algorithm, we can efficiently solve the original pattern discovery problem with several indexing and searching techniques. Comprehensive experiments on sequence pattern analysis and a study on cancer genomics application demonstrate the effectiveness and efficiency of our algorithm, compared to several existing methods.

1 Introduction

Consecutive pattern mining, also known as *string pattern mining*, is one of the most important problem in the field of frequent pattern mining [4, 3, 6, 21, 7, 4]. Briefly, given a set of data sequences, the problem is to discover subsequences that are frequent, i.e. occurring more than or equal to a minimum support threshold (σ), where σ is a user-defined parameter. Mining frequent consecutive patterns has been widely applied in many real-world problems, including those in bioinformatics, time series analysis, and computer networks. Currently, mining consecutive repeats (i.e., frequent consecutive patterns) is one of the most important methods and techniques in bioinformatics for studying functions of biological molecules, including DNA, RNA, and proteins. Similar molecular subsequence motifs often imply specific biological functions. As the biological data repositories growing explosively in recent years, efficient algorithms to identify string patterns becomes a pressing need. String data structures [31, 15] have been utilized extensively to find all of the exact consecutive repeats and patterns in both linear time and space complexity. However, these methods only take care of exactly matched repeats and patterns; but in practice, the repeats and patterns may differ slightly in different occurrences. For example, even today's most advanced DNA sequencing machines would produce sequencing errors with a small yet non-negligable rate. In addition, the mutational process in organisms can introduce mutations or shifts on random locations in DNA, RNA or protein sequences [21]. Similarly, in network analysis, the malicious behaviors, even from the same malware program, could appear slightly different among payload traffic logs [32]. Therefore, mining consecutive patterns is not a well-solved problem, as shown in Example 1.

Example 1. ***(Why Approximate and Edit Distance?)*** *Let the dataset contain the following three DNA sequences and the minimum support threshold (σ) be 3. First of all, they are not same only due to a small amount of differences. Even by considering Hamming Distance ≤ 1, none of them will be treated as a frequent pattern since $\underline{C}$ is missing in the third sequence.*

$$\ldots AC\underline{C}GT\underline{G}TAGGTCG\ldots$$
$$\ldots AC\underline{C}GT\underline{T}TAGGTCG\ldots$$
$$\ldots AC\underline{\ }GT\underline{G}TAGGTCG\ldots$$

However, if we formulate the matching by Edit Distance (which will make the problem harder than Hamming Distance), the three sequences above can be discovered as frequent patterns. This will be very useful for DNA *analysis where insertions and deletions are very common.*

Considering that patterns, especially long ones, may occur multiple times with slight differences, there is a need to formulate a new frequent *approximate pattern* mining problem, by defining approximate supports. Previous research on finding DNA repeats (i.e., frequent consecutive patterns when $\sigma = 2$) mainly choose *Hamming Distance* to define the approximate matching. REPuter [18] is the closest effort toward mining frequent approximate consecutive patterns under *Hamming Distance*. Recently, some novel and special rules and Trie-based data structure is also proposed to further improve the efficiency [26]. However, they can only discover patterns with two occurrences and mismatches at identical positions across the support set. It is hard to extend them from $\sigma = 2$ to general σ values. Moreover, *Hamming Distance* cannot capture the differences of insertions and deletions, as shown in Example 1. Therefore, it is necessary to develop a novel and efficient algorithm for general settings under a new distance measure, such as *Edit Distance*.

Due to the downward closure property (i.e., any subset of a frequent pattern is also frequent), there exist an explosive number of frequent patterns. Both closed and maximal patterns are compressed patterns. But for approximate pattern mining, such as DNA sequence analysis, maximal pattern is more concise since each maximal may contain many shorter closed patterns with different support frequencies. Nevertheless, despite that one can apply an exhaustive search algorithm on small scale datasets, the computational cost for finding such patterns on large datasets, e.g., human genome with billions of base pairs, is still unacceptable.

Based on the above discussion, we formulate the problem of mining *Maximal Approximate Consecutive Frequent Patterns* (MACFP) and provide an efficient and scalable solution. The main contributions of this study are as follows.

- This is the first study (to the best of our knowledge) that formulates the general problem of *Maximal Approximate Consecutive Frequent Pattern Mining* (MACFP) using edit distance.
- Several key properties are discovered and proved, and an efficient algorithm is developed to find out all maximal approximate consecutive frequent patterns.
- We provide a theoretical analysis of its time complexity and then empirically demonstrate the effectiveness and efficiency of the proposed algorithm on different datasets.
- We demonstrate the practical usage of the algorithm at detecting short tandem repeats and copy number variations in cancer genomics.

2 Related Work

Mining Repeats in Bioinformatics. Repeat-related problems are well studied in the fields of bioinformatics [1]. Finding tandem repeats, i.e., the substrings (approximately) repeated at least k times consecutively, is firstly proposed and resolved in the form of exact matching [5]. Approximate tandem repeats problems under Edit Distance or Hamming Distance is solved and improved by divide-and-conquer algorithms utilizing the property of consecutive occurrences [17, 28]. However, as long as the occurrences of patterns are not restricted to be adjacent, the previous tools lose their key property for acceleration and thus become inefficient. Discovering all exact matched repeats can be perfectly solved by Suffix Tree [25]. For those approximate repeats algorithms [18, 26], although they are efficient when $\sigma = 2$, they are specially designed for “at least twice”. For example, the extension techniques in REPuter [18] require mismatches are at identical positions across the support set. Therefore these methods are difficult to be tailored in our settings due to the general support threshold.

General Pattern Mining. Consecutive frequent patterns take crucial roles in many fields, such as periodic patterns in temporal data sequence [14], clicking patterns in a large web database [10], patterns like copy-paste bugs in software [22], and closed repetitive gapped sequential patterns [8]. Apriori-based approaches [29, 27] typically fail to attack this problem due to the exponential number of intermediate patterns (itemsets). Pattern fusion algorithm [34] generates larger itemsets by merging smaller frequent itemsets with similar numbers of occurrences, such that many of the medium-size frequent itemsets are skipped and thus be efficient. However, itemset is a little different from pattern, especially when assembling. We can simply use the set union to assemble different small itemsets, but the order of small patterns should be considered when merge. Therefore, the problem of mining long frequent consecutive patterns faces greater challenges. Approximate pattern mining under Hamming Distance is addressed by aligning the long patterns by small chunks [33]. We are also using the ideas of small chunks, however, due to allowing insertions and deletions in Edit Distance, directly applying the original alignment algorithm becomes almost impossible. String similarity search supporting edit distance [23] is also related to our problem. However, it is focused on short strings and the number of its candidate strings is much smaller than ours.

3 Preliminaries

3.1 Notations Let S be a string of length $|S| = n$. S_i denotes the i-th character of S. The alphabets set of S, i.e., $\{c|\exists i, S_i = c\}$, is represented by Σ (e.g., for DNA sequences, $\Sigma = \{A, C, G, T\}$). For any pair (i, j), where $i \leq j$, we can get a substring $S_{i,j}$ from i-th character to j-th character (both inclusive).

For two substrings $S_{i,j}$ and $S_{x,y}$, we define their distance by the Edit Distance. That is, $d(S_{i,j}, S_{x,y})$ describes the minimum number of edit operations (insertion, deletion, and substitution) needed to transform string $S_{i,j}$ to string $S_{x,y}$.

3.2 Problem Formulation We first define equivalence between two substrings.

Definition 1. **(Approximate) Equivalent Neighbors**. *Two substrings $S_{i,j}$ and $S_{x,y}$ are* (approximate) equivalent neighbors *if and only if their edit distance $d(S_{i,j}, S_{x,y})$ is no more than a given distance threshold k, i.e., $d(S_{i,j}, S_{x,y}) \leq k$.*

In the real world application, such as DNA pattern mining, the distance threshold k is typically small, because people are only interested in those slightly different DNA substrings (e.g., at the length of $O(\log n)$).

Example 2. *Given $k = 2$, ACGACA and ACGTACG are approximate equivalent, because their distance is 2. However, AACCGA and ACCAAG are not, since their distance is 4.*

The simple counting of the equivalent occurrences as the support will not work properly in the approximate scenario. For example, when $k = 3$, there are many trivial approximate neighbors for substring $S_{i,j}$, such as $S_{i-3,j}$, $S_{i-2,j}$, $S_{i-1,j}$, $S_{i+1,j}$, $S_{i+2,j}$, $S_{i+3,j}$, $S_{i,j-3}$, $S_{i,j-2}$, $S_{i,j-1}$, $S_{i,j+1}$, $S_{i,j+2}$, $S_{i,j+3}$, etc. This kind of trivial approximate neighbors makes the simple counting becomes large and meaningless. Therefore, for a given substring $S_{i,j}$, we propose the following formal definition for its approximate support.

Definition 2. **Approximate Support** *$sup(S_{i,j})$. The approximate support of $S_{i,j}$ is the maximum number p, such that there exist p **disjoint** equivalent neighbors. That is, there exist*

$$l_1 \leq r_1 < l_2 \leq r_2 < \ldots < l_x = i \leq r_x = j < \ldots < l_p \leq r_p$$

where, $\forall q (1 \leq q \leq p), d(S_{i,j}, S_{l_q, r_q}) \leq k$.

Similar ideas of using maximal number of non-overlapping windows as a support for sequential patterns have also been suggested in [19].

Example 3. *Given a string*

$$S = AGCTAGCAGAGCT$$

and the distance threshold $k = 1$, the $sup(S_{1,4} = AGCT) = 3$. One possible plan is shown below.

$$\lceil AGCT \rfloor \lceil AGCA \rfloor G \lceil AGC \rfloor A$$

where, $\lceil \rfloor$ highlights the approximate neighbors of $AGCT$.

Definition 3. **Approximate Consecutive Frequent Pattern (ACFP)**. *A substring $S_{i,j}$ is called Approximate Consecutive Frequent Pattern, if and only if $sup(S_{i,j})$ is no less than a given threshold σ, i.e. $sup(S_{i,j}) \geq \sigma$.*

It is worth noting that, in previous research of DNA repeats (i.e. $\sigma = 2$), they also require the repeat should be somewhere else, which is also consistent with our disjoint requirement.

Example 4. *Based on the previous example, if $\sigma = 3$, $S_{1,4} = AGCT$ is an ACFP, because $sup(S_{1,4}) = 3 \geq \sigma$. However, $S_{1,5} = AGCTA$ is **not** an ACFP, because $sup(S_{1,5}) = 2 < \sigma$.*

Definition 4. **Maximal Approximate Consecutive Frequent Pattern (MACFP)**. *An ACFP $S_{i,j}$ is called MACFP, if and only if*

- *$|S_{i,j}| \geq L$, i.e. it is long enough.*
- *$S_{i-1,j}$ and $S_{i,j+1}$ are not ACFP.*

Here, L is a length threshold to focus on long patterns.

Example 5. *Based on the previous example, if $L = 3$, $S_{1,4} = AGCT$ is a MACFP, because it is a long-enough ACFP; $S_{1,5}$ is not an ACFP, and $S_{0,4}$ is not a valid substring.*

Our **objective** is to find all MACFP, given string s, edit distance threshold k, minimum support threshold σ, and length threshold L. To the best of our knowledge, this is the first work that formulates and attacks such a general maximal approximate consecutive frequent pattern mining (MACFP) problem.

3.3 Exact Consecutive Pattern Mining Exact consecutive pattern mining, as a special case of our problem ($k = 0$), is well studied before. In this paper, we directly use the state-of-the-art algorithms as our weapons to resolve the more general problem. We utilize the exact pattern mining algorithm with linear time and memory, which takes the advantage of indexing all suffixes of S by Suffix Array [15]. Techniques like suffix ordering, the longest common prefix (LCP) intervals [2] and range minimum query (RMQ) [9] provide great help for our further approximate pattern mining, by supporting $O(1)$ longest common prefix query for any pair of $S_{i,n}$ and $S_{j,n}$ after $O(n)$ pre-computation.

4 Methodology

We first propose a novel and efficient framework to reduce the number of support checking queries from $O(n^2)$ to $O(n)$ and also demonstrate its necessity. Focusing on the efficient calculation of the approximate support, by discovering several important properties, novel techniques are developed, such as efficient expanding, lower bound pruning, and fast chunk indexing.

4.1 Support Checking Framework In order to get rid of tremendous medium-length patterns, we propose to attack the MACFP mining problem via a novel angle different from traditional pattern growth ideas. The new idea is that reducing the MACFP mining problem into the decision problem: checking the approximate support of each substring. More specifically, we assume that there is an oracle which can instantly tell us the approximate support of any substring $S_{i,j}$ and we try to minimize the number of times to query the oracle. The efficient calculation of approximate support of substrings will be discussed in later sections.

Based on the definition of p_i, we can easily develop a bruteforce algorithm, which checks the supports of all substring of $S_{i,j}$ and then figures out all MACFPs. In the worst cases, the number of times to check the support of some substring is $O(n^2)$. However, there is still some room to improve it since we actually have only $O(n)$ MACFPs.

We first have Lemma 1 as following, which is quite intuitive (see proof in supplementary material[1]).

LEMMA 1. ***Pattern Anti-Monotonicity****, If $S_{i,j}$ is a approximate consecutive frequent pattern, so are $S_{i+1,j}$ and $S_{i,j-1}$.*

DEFINITION 5. **Extremal Pattern Position** p_i, *which is the farthest position j such that $S_{i,j}$ is an approximate consecutive frequent pattern, that is,*

$$p_i = \max\{j | sup(S_{i,j}) \geq \sigma\} \tag{4.1}$$

Note that "extremal" is different from "maximal", because the left side of S_{i,p_i} still have chances to be extended further.

EXAMPLE 6. $p_1 = 4$ *when* $S = AGCTAGCAGAGCA$, $k = 1$ *and* $\sigma = 3$.

LEMMA 2. ***Monotonicity of*** p_i.

$$\forall 2 \leq i \leq n, p_i \geq p_{i-1} \tag{4.2}$$

Algorithm 1: Support Checking Framework

```
Require: A string $S$, parameters $k, \sigma$
Return: $\forall i \leq n$, extremal pattern positions $p_i$
Time Complexity: $O(n)$ times of support
checking
$p_0 \leftarrow 0$
for $i = 1$ to $n$ do
    $j \leftarrow p_{i-1}$
    while $j + 1 \leq n$ and $sup(S_{i,j+1}) \geq \sigma$ do
        $j \leftarrow j + 1$
    $p_i \leftarrow j$
return $p$
```

Proof. $S_{i-1,p_{i-1}}$ is an approximate consecutive frequent pattern, implies $sup(S_{i,p_{i-1}}) \geq \sigma$, and thus $p_i \geq p_{i-1}$.

Utilizing the monotonicity of p_i, we optimize the number of times of support checking from $O(n^2)$ to $O(n)$ as shown in Algorithm 1. An important observation is that both i and j are non-decreasing all the time and after each support checking of the substring $S_{i,j}$, at least one of them is increased by 1. Therefore, the total times of support checking is at most $2n$, which is $O(n)$. Meanwhile, there are at most $O(n)$ MACFPs in total, which makes $O(n)$ times of support checking necessary.

4.2 Efficient Support Computation We propose an efficient algorithm to compute the approximate support for any substring $S_{i,j}$ in the expectation time complexity $O(k3^k \frac{n}{|\Sigma|^{\lfloor \frac{L}{k+1} \rfloor}})$.

4.2.1 From Neighbors to Non-overlapping Neighbors As mentioned in the problem formulation, we define the maximum number of disjoint (non-overlapping) equivalent neighbors as our approximate support. It is hard to directly find and only find those equivalent neighbors in the maximum disjoint solution. Therefore, we can find a set of "necessary" equivalent neighbors and then compute the approximate support, which is a classical interval scheduling problem. Lemma 3 (proof in supplementary file) allows us to focus on these extremely small equivalent neighbors when calculating the approximate support, which helps us greatly reduce the number of "necessary" equivalent neighbors.

LEMMA 3. *Suppose $S_{u+1,v}$ or $S_{u,v-1}$ is an equivalent neighbor of $S_{i,j}$, the $sup(S_{i,j})$ is same if we ignore $S_{u,v}$.*

4.2.2 Efficient Expanding To compute the edit distance, dynamic programming algorithm avoids exponential searches by recording all previous states, while the naive way is to enumerate all possible operations at those

[1] http://web.engr.illinois.edu/~shang7/papers/MACFP-Supplementary.pdf

Algorithm 2: Efficient Expanding

```
Require: a substring $S_{l,r}$, a suffix $S_{v,n}$, and distance threshold $k$
Return: $\forall i \leq k$, $q_i = \min\{j | d(S_{l,r}, S_{v,j}) \leq i\}$
Time Complexity: $O(3^k)$
vector $q \leftarrow +\infty$, $C \leftarrow \{(l, v, 0)\}$
while $|C| > 0$ do
    Get a triplet $(x, y, d)$ and remove it from $C$
    if $x = r + 1$ then
        $q_d \leftarrow \min\{q_d, y\}$
        continue
    if $S_x = S_y$ then
        $\delta \leftarrow \min\{lcp(S_{x,n}, S_{y,n}), r - x + 1\}$
        $C \leftarrow C \cup \{(x + \delta, y + \delta, d)\}$
    else
        for $\delta_x = 0$ to $k - d$ do
            for $\delta_y = 0$ to $k - d$ do
                if $x + \delta_x = r + 1$ or $S_{x+\delta_x} = S_{y+\delta_y}$ then
                    $C \leftarrow C \cup \{(x + \delta_x, y + \delta_y, d + \max(\delta_x, \delta_y))\}$
for $i = 0$ to $k - 1$ do
    $q_{i+1} \leftarrow \min\{q_i, q_i - 1\}$
return vector $q$.
```

points and try all possible combinations, which is $O(3^k)$. However, in our problem settings, there is an edit distance threshold k which is typically $O(\log L)$ and thus $O(3^k)$ is $O(L^{c \log 3})$ which is cheap. Here c is a constant coming from that $k \leq c \log L$.

Lemma 4. ***Greedy Matching.*** *For two strings s and t, if $s_1 = t_1$, it is better to match them when calculating edit distance between strings s and t. That is, $d(s, t) = d(s_{2,|s|}, t_{2,|t|})$ if $s_1 = t_1$.*

Proof. This property is utilized in the traditional dynamic programming algorithm for computing edit distance [24]. It also fits human intuition that the matches in both ends are good to use rather than to edit.

Moreover, based on Lemma 4, we can also make sure that after several consecutive operations, the two new positions should be matched each other. Therefore, an efficient expanding algorithm, as shown in Algorithm 2, provides another choice for us to replace the dynamic programming and get rid of the length of string $|s|$ in the time complexity. This improvement will be significant when the length of string becomes long.

4.2.3 Fast Chunk Indexing

Lemma 5. ***Exact Chunks.*** *After segmenting $S_{i,j}$ into $k + 1$ chunks, for any equivalent neighbor of $S_{i,j}$, there exists an exact matched chunk.*

Algorithm 3: Fast Chunk Indexing

```
Require: a substring $S_{l,r}$, distance threshold $k$
Return: its approximate support
Time Complexity: expected $O(k3^k \frac{n}{|\Sigma|^{\lfloor \frac{r-l+1}{k+1} \rfloor}})$
$C \leftarrow \emptyset$, $\epsilon \leftarrow \lfloor \frac{r-l+1}{k+1} \rfloor$
for $i = 1$ to $k + 1$ do
    $s \leftarrow l + i \cdot \epsilon$, $t \leftarrow l + (i + 1) \cdot \epsilon - 1$
    // exact occurrences querying
    for all $S_{u,v} = S_{s,t}$ do
        $d_1 \leftarrow$ lower bound between $S_{*,u}$ and $S_{l,s}$
        $d_2 \leftarrow$ lower bound between $S_{v,*}$ and $S_{t,r}$
        // lower bound pruning
        if $d_1 + d_2 > k$ then
            continue
        // call efficient expanding twice
        $ql_x \leftarrow \max\{j | d(S_{l,u+1}, S_{j,s-1}) \leq x\}$
        $qr_y \leftarrow \min\{j | d(S_{v+1,r}, S_{t+1,j}) \leq y\}$
        for $x = 0$ to $k$ do
            for $y = 0$ to $k - x$ do
                $C \leftarrow C \cup \{(ql_x, qr_y)\}$
return maximum non-overlap intervals in $C$
```

Proof. Because the edit distance threshold is k and the Pigeonhole principle, there is at least one chunk that is exactly matched.

As shown in Algorithm 3, it finally makes the expected time complexity of our method in $O(nk3^k \frac{n}{|\Sigma|^{\lfloor \frac{L}{k+1} \rfloor}})$ by incorporating the support checking framework and efficient expanding algorithms together. The analysis goes as following: 1) the size of each chunk is at least $O(\lfloor \frac{L}{k+1} \rfloor)$; 2) there are $O(|\Sigma|^{\lfloor \frac{L}{k+1} \rfloor})$ distinct chunks in total; 3) there are $O(n)$ length-L substrings in S; 4) assuming the string is randomly generated, the expectation of occurrences of a chunk is $\frac{n}{|\Sigma|^{\lfloor \frac{L}{k+1} \rfloor}}$. Moreover, the time complexity becomes $O(nL^{c \log L} \log L \frac{n}{|\Sigma|^{\lfloor \frac{L}{k+1} \rfloor}})$ when k is $O(\log L)$. Thus it is really efficient especially for large L.

4.2.4 Lower Bound Pruning

Lower bound pruning strategies are usually used in q-gram approximate string processing [13]. In our paper, we also propose a similar lower bound of edit distance as following.

Lemma 6. ***Lower Bound of Edit Distance.***

$$d(s, t) \geq \max\{v_{s,t}, v_{t,s}\} \tag{4.3}$$

where,

$$v_{s,t} = \sum_{c \in \Sigma} \max\{count_c(s) - count_c(t), 0\} \tag{4.4}$$

and $count_c(s)$ is the number of character c in string s.

Proof. When proving the lower bound, we can assume all the same characters match each other in strings s and t, which is the best case. The minimum cost is $\max\{v_{s,t}, v_{t,s}\}$ in this case.

EXAMPLE 7. *Given two strings $s = ACAGGA$ and $t = ACGGTT$. There are 3 A's, 1 C, and 2 G's in s. And t has 1 A, 1 C, 2 G's, and 2 T's. Based on Lemma 6, we can compute $v_{s,t} = 2$ and $v_{t,s} = 2$. Therefore, the lower bound is 2.*

Using this lower bound, we can prune out some trivial non-similar substrings before the efficient expanding algorithm is performed, which makes our method much more efficient. More specifically, given a substring $s_{l,r}$, when we are considering the potential equivalent neighbors among the substring starting from index u, i.e., the substrings $s_{u,*}$, we can calculate a lower bound of distances to pre-check whether we need to run the efficient expanding algorithm to do further checking (algorithm in supplementary file).

5 Experiments

We evaluate the performance of our proposed approach and explore the speedup techniques used in the approach under different parameter settings.

5.1 Datasets DNA sequence is one of application scenarios of our method. Therefore, to evaluate the run time of our approach on the data of various sizes, we select the following 3 datasets with different DNA string lengths n from **Human Genome DNA Sequence.** The alphabet set for DNA sequence is $\Sigma = \{A, C, G, T\}$.

- Small Dataset, $n =$**10K**, culled from chr1;
- Medium Dataset, $n =$**100K**, culled from chr1;
- Large Dataset, $n =$**1M**, culled from chr6.

As analyzed before, our proposed method MACFP will be faster when the alphabet set Σ is larger, due to our pruning and indexing techniques. As a result, DNA sequences consisting of only 4 different symbols reflect almost the worst case of MACFP in running time.

5.2 Experiment Setting All experiments are performed on a local machine with Intel(R) Xeon(R) CPU E3-1240 @3.4GHz and 8GB memory. Although many parts of our algorithms could be parallelized, in the purpose of evaluating the algorithmic efficiency, we only allow a single thread during experiments.

We have 4 methods as introduced below.

- **TDP** finds all approximate neighbors of $S_{l,r}$ starting from i-th position via a widely used dynamic programming [30, 16] in $O((r-l)k)$ time. Fitting into our linear support checking framework, its worst case time complexity is $O(n^3k)$, while the best is $O(n^2Lk)$. Due to its expensive running time, we only run it on small dataset. See details in supplementary material.
- **TDP+** applies Fast Chunk Indexing technique to accelerate TDP.
- **MACFP−** turns off Lower Bound Pruning technique in MACFP.
- **MACFP** is our proposed algorithm.

Table 1: Values for different parameters.

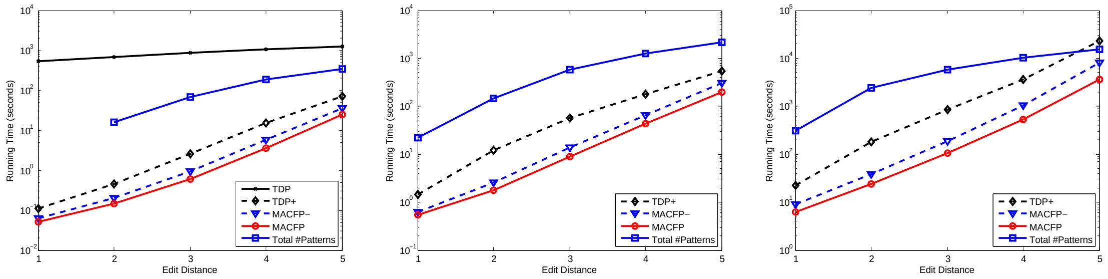
Parameter	Values (bold is default)
Edit Distance Threshold k	1, 2, **3**, 4, 5
Length Threshold L	30, 40, **50**, 60, 70
Minimum Support σ	2, 3, **4**, 5, 6

Table 1 shows the detail of the parameters. Every time, we will change a single parameter and fix others to their default values. On the Small dataset, there will be no pattern if L is larger than 50. Therefore, on the Small dataset, $L \in \{10, 20, 30, 40, 50\}$, and $L = 30$ is the default parameter for the Small dataset.

5.3 Impact of Different Techniques As shown in Figure 1, Figure 2, and Figure 3, although the run time of TDP seems linear to the edit distance k, it is still the slowest. Fast Chunk Indexing makes TDP+ a significant improvement over TDP, because many irrelevant positions get pruned before dynamic programming. The advantage of MACFP− over TDP+ shows that the Efficient Expanding is very effective, especially for long patterns (i.e. larger L). The Lower Bound Pruning strategy demonstrates its power by reducing the running time from MACFP− to MACFP.

5.4 Impact of Parameters Edit Distance Threshold. As verified in Figure 1, the running time of our method is exponential to k. However, MACFP still follows (even slower) the trend of the growth of the number of total patterns, which is very interesting but still fits human intuition because a larger k provides more flexibility for DNA patterns such that more patterns could be detected. That means, MACFP only pays more efforts to get those patterns, which could be the perfect situation.

Length Threshold. Experimental results shown in Figure 2 demonstrate the running time of our method is inversely exponential to L. That is, when L grows, the running time of MACFP dramatically decreases in all three different sizes of datasets, which agrees with our theoretical analysis. This is exciting because scientists may only concern about the long and abnormal patterns rather than short and common patterns. It is worth noting that the advantage of MACFP− over TDP+ on

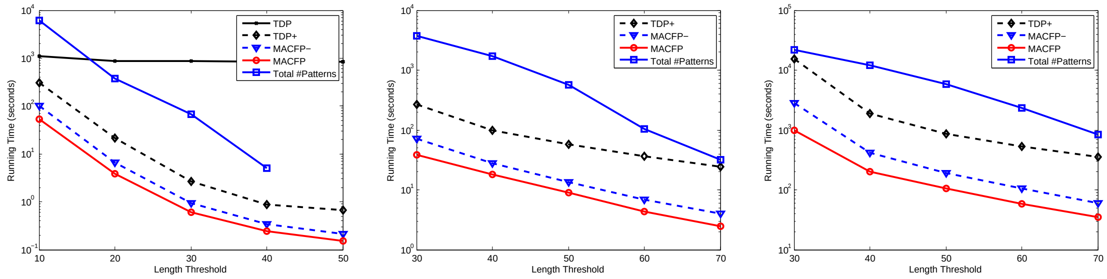


(a) small dataset

(b) medium dataset

(c) large dataset

Figure 1: Varying edit distance. The total number of patterns is 0 on small dataset when $k = 1$.



(a) small dataset

(b) medium dataset

(c) large dataset

Figure 2: Varying length threshold. The total number of patterns is 0 on small dataset when $L = 50$.

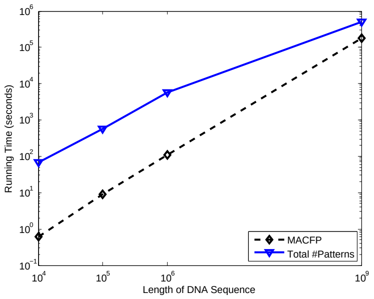


Figure 4: Varying the length of the sequence.

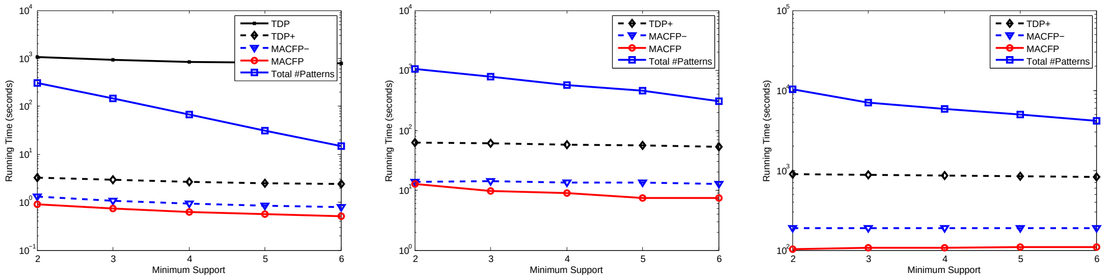
the large dataset is very big when the length threshold is very small. In this case, the chunk is so short that the portions of patterns remained to check are large, which implies Efficient Expanding is much faster than Tailored Dynamic Programming.

Minimum Support Threshold. The theoretical time complexity is not directly related to σ, because we are using the novel support checking framework which gets rid of pattern growth procedures. Therefore, we only vary its values between 2 and 6, aiming to explore the performance when even infrequent patterns appear in the dataset. As shown in Figure 3, the running times except for TDP are almost same when σ varies, which is consistent with our analysis.

5.5 Scalability By concatenating DNA sequences from different human genes, we finally assembled a billion length dataset. By setting $\sigma = 4, k = 3, L = 180$, MACFP outputs $505,362$ patterns using about 50 hours. As shown in Figure 4, by plotting this and other default settings' results, one can observe that the running time is always following the number of patterns and are log-linear to the length of the sequence. Moreover, the processing can be parallelized and thus should be a useful approach for billion length sequence. We will report additional performance results with varied parameters for this extremely large dataset in the revised version, after more extensive performance study is conducted.

6 Applications

In many cancer studies, people have found the DNA sequences of patients have abnormally highly repetitive regions, which were not found in normal people's DNAs. Specifically, there are two categories of such observations: (i) *short tandem repeat*, in which there exists a short piece of normal DNA sequence fragment, consecutively repeated with a large number of copies and inserted at the same location where the normal substring occurs, and (ii) *copy number variation*, where the repeated fragment is much longer while the times it gets repeated is smaller. Further, in practice, it is impossible to get complete DNA sequences from patients using existing technologies. Instead, scientists use sequencing machines which first cut the full-length DNA sequences into short pieces or reads and then using chemical methods to readout



(a) small dataset

(b) medium dataset

(c) large dataset

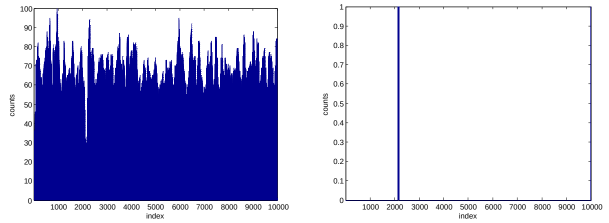
Figure 3: Varying minimum support.

the content of these short reads. A reasonable length of these short reads is 100 which is the length of reads generated from the most widely used Illumina sequencing machine. Meanwhile, the DNA sequences obtained from sequencing machines are always noisy such that *exact* repeats are unsuitable for these problems. To evaluate our method in these real-world scenarios, we design a synthetic experiment as follows.

1. Select a length-n sequence S at the beginning of chr1 of human as the *normal* DNA sequence.
2. Sample a length-m subsequence s with relatively high entropy from S as the *fatal* subsequence.
3. Duplicate s for T times. We allow at most 1 edit distance (10% probability per edit type) for potential variation in each copy. The new (*patient*) DNA sequence is denoted by P.
4. Random access length-100 subsequence n times from the patient DNA sequence P. Concatenate them together with the normal DNA sequence S to get a long (about $n \times 100$) DNA sequence for diagnose.

Traditionally, the *read mapping and counting* (denoted as **RMC** in this paper) technique is widely used, which uses matching algorithms, such as BWA [20] and mrsFAST [12, 11] to find all the matched positions of the gene fragments on the normal DNA sequence S, and then visualize these matches and focus on the most heated regions. However, due to the self-similarity inside the reference sequence and also the sequencing errors of reads, the result is always not optimal and often misses the correct region. Now scientists can run the MACFP algorithm to efficiently figure out the potential fatal region. We choose different parameters according to the *short tandem repeat* ($n = 10,000$, $m = 50$, and $T = 100$) and *copy number variation* ($n = 10,000$, $m = 1,000$, and $T = 20$) problems.

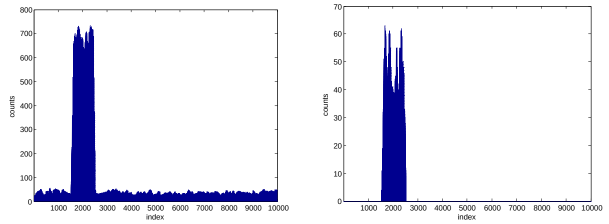
As shown in Figure 5 and Figure 6, MACFP can always identify the repetitive DNA region more correctly and clearly. However, in Figure 5, RMC finds several peaks and the highest one is around 1,000 while the correct one is actually about the third or fourth peak. Therefore, RMC may even mislead scientists in this case.



(a) RMC

(b) MACFP

Figure 5: Short tandem repeats. The repetitive region is [2125, 2174]. MACFP: $k = 1$, $\sigma = 500$, $L = 50$



(a) RMC

(b) MACFP

Figure 6: Copy number variation. The repetitive region is [1535, 2534]. MACFP: $k = 1$, $\sigma = 20$, $L = 100$

7 Conclusions

In this paper, we formulated the general problem of *maximal approximate consecutive frequent pattern mining* using edit distance, and developed a novel algorithm *MACFP* that efficiently solves this problem. Comprehensive experimental results demonstrated the effectiveness and efficiency of our algorithm compared to existing methods. Furthermore, we have successfully applied our algorithm on two important applications in genomics, with significant improvements over the widely used bioinformatics method. In future, we plan to develop readily usable tools for DNA pattern mining problems by further considering reverse complementation and handling paired-end reads.

Acknowledgements
Research was sponsored in part by the U.S. Army Research Lab. under Cooperative Agreement No. W911NF-

09-2-0053 (NSCTA), National Science Foundation IIS-1017362, IIS-1320617, and IIS-1354329, HDTRA1-10-1-0120, and grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov), and MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC.

References

[1] M. Abouelhoda and M. Ghanem. String mining in bioinformatics. In *Scientific Data Mining and Knowledge Discovery*, pages 207–247. Springer, 2010.

[2] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[3] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, volume 22, pages 207–216. ACM, 1993.

[4] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, volume 1215, pages 487–499, 1994.

[5] G. Benson. Tandem repeats finder: a program to analyze dna sequences. *Nucleic acids research*, 1999.

[6] H. Cao, N. Mamoulis, and D. W. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, pages 8–pp. IEEE, 2005.

[7] X. Chen, S. Kwong, and M. Li. A compression algorithm for dna sequences and its applications in genome comparison. In *Computational molecular biology*, page 107. ACM, 2000.

[8] B. Ding, D. Lo, J. Han, and S.-C. Khoo. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *ICDE*, pages 1024–1035. IEEE, 2009.

[9] J. Fischer and V. Heun. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Combinatorial Pattern Matching*, pages 36–48. Springer, 2006.

[10] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.

[11] F. Hach, F. Hormozdiari, C. Alkan, F. Hormozdiari, I. Birol, E. E. Eichler, and S. C. Sahinalp. mrsfast: a cache-oblivious algorithm for short-read mapping. *Nature methods*, 7(8):576–577, 2010.

[12] F. Hach, I. Sarrafi, F. Hormozdiari, C. Alkan, E. E. Eichler, and S. C. Sahinalp. mrsfast-ultra: a compact, snp-aware mapper for high performance sequencing applications. *Nucleic acids research*, page gku370, 2014.

[13] M. Hadjieleftheriou and D. Srivastava. *Approximate string processing*. Now Publishers Inc, 2011.

[14] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *ICDE*, pages 106–115. IEEE, 1999.

[15] J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In *Automata, Languages and Programming*, pages 943–955. Springer, 2003.

[16] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[17] R. Kolpakov, G. Bana, and G. Kucherov. mreps: efficient and flexible detection of tandem repeats in dna. *Nucleic acids research*, 31(13):3672–3678, 2003.

[18] S. Kurtz, J. V. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. Reputer: the manifold applications of repeat analysis on a genomic scale. *Nucleic acids research*, 29(22):4633–4642, 2001.

[19] S. Laxman, P. Sastry, and K. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *SIGKDD*, pages 410–419. ACM, 2007.

[20] H. Li and R. Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[21] J. Li, R. Lupat, K. C. Amarasinghe, E. R. Thompson, M. A. Doyle, G. L. Ryland, R. W. Tothill, S. K. Halgamuge, I. G. Campbell, and K. L. Gorringe. Contra: copy number analysis for targeted resequencing. *Bioinformatics*, 28(10):1307–1313, 2012.

[22] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *Software Engineering, IEEE Transactions on*, 32(3):176–192, 2006.

[23] W. Lu, X. Du, M. Hadjieleftheriou, and B. C. Ooi. Efficiently supporting edit distance based string similarity search using b-trees. *TKDE*, 26(12):2983–2996, 2014.

[24] J. H. Martin and D. Jurafsky. Speech and language processing. *International Edition*, pages 107–111, 2000.

[25] E. M. McCreight. A space-economical suffix tree construction algorithm. *JACM*, 23(2):262–272, 1976.

[26] S. Pal and S. Rajasekaran. Improved algorithms for finding edit distance based motifs. *bioRxiv*, 2015.

[27] 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*, page 0215. IEEE Computer Society, 2001.

[28] D. Sokol, G. Benson, and J. Tojeira. Tandem repeats over the edit distance. *Bioinformatics*, 23(2):e30–e35, 2007.

[29] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *KDD*, volume 97, pages 67–73, 1997.

[30] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684. IEEE, 2002.

[31] P. Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory*. IEEE, 1973.

[32] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *DIMVA*, pages 207–227, Berlin, Heidelberg, 2008. Springer-Verlag.

[33] F. Zhu, X. Yan, J. Han, and P. S. Yu. Mining frequent approximate sequential patterns. *Next Generation of Data Mining*, pages 66–87, 2009.

[34] F. Zhu, X. Yan, J. Han, P. S. Yu, and H. Cheng. Mining colossal frequent patterns by core pattern fusion. In *ICDE*, pages 706–715. IEEE, 2007.