An Efficient Algorithm for Finding Long Conserved Regions Between Genes

Tak-Man Ma[1], Yuh-Dauh Lyuu[2,*], and Yen-Wu Ti[2]

[1] Dept. of Computer and Information Science,
University of Pennsylvania, Philadelphia, USA
mata3@seas.upenn.edu
[2] Dept. of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
{lyuu, d91010}@csie.ntu.edu.tw

Abstract. We study the problem of approximate non-tandem repeat (conserved regions) extraction among strings (genes). Basically, given a string S and thresholds L and D over a finite alphabet, extracting approximate repeats is to find pairs (β, β') of substrings of S under some constraints such that β and β' have edit-distance at most D and their respective lengths are at least L. Previous works mainly focus on the case that D is small, so they are not appropriate for extracting approximate repeats with relatively large D. In contrast, this paper focuses on extracting long approximate repeats with large D and it is more efficient than previous works. We also show that our algorithm is optimal in time when D is a constant.

In this paper, given an input string S and thresholds L and D, we would like to extract all (D, L)-supermaximal approximate repeats (β, β') of S. One useful application of extracting all (D, L)-supermaximal approximate repeats (β, β') is to find all longest possible substrings β of S such that there exist some other substring β' of S where β and β' have edit-distance at most D and their respective lengths are at least L. This algorithm can be easily applied to the case where there are multiple input strings $S_1, S_2, \ldots, S_n$ if we first concatenate the input strings into one long subject string S with a special symbol "$\sharp$" for separation: $S_1\sharp S_2\sharp \ldots \sharp S_n$. The running time complexity of our algorithm is $O(DN^2)$ where $N = \mid S_1 \mid + \mid S_2 \mid + \cdots + \mid S_n \mid$.

1 Introduction

1.1 Related Work

The repetitive structure of genomic DNA holds many secrets that await discovery. Conserved regions between genes have important meanings in biology and play an important role in the identification of novel functional units [4]. For example, the gene sequences of a gene family may share conserved regions that

* Corresponding author.

correspond to meaningful domains in their protein structures. A useful approach for identifying meaningful conserved regions is to find non-tandem approximate repeating patterns that occur more frequently than expected by chance. The problem of exact repeat extraction seems settled since it already has an optimal linear-time algorithm by suffix tree [4]. However, as mutations often render DNA copies imperfect, it is much more important to study approximate repeats.

The first paper that dealt with model-based recognition of degenerate repeats solved the problem of finding the highest-scored pair of (possibly overlapping) substrings in a string in $O(N^2)$ time and space, where N is the length of input string [3]. Kannan and Myers [5] and Benson [2] restrict the outputs to pairs of non-overlapping substrings. Both algorithms run in $O(N^2 \log^2 N)$ time. The space usage is $O(N^2 \log N)$ [5], which was later improved to $O(N^2)$ [2]. A related question is to find all approximate repeats of a string which are within a fixed edit-distance threshold, which is the main focus of this paper. A detailed survey of previous work on finding approximate repeats can be found in [6].

A popular idea for finding maximal approximate repeats (as defined later in Section 1.2) which are within a fixed edit-distance threshold is to first search for small exact repeats as seed strings and then form maximal approximate repeats by extension. Kurtz et al. [6] have presented one such algorithm for finding maximal approximate repeats of length L which runs in $O(N + D^3 z)$ by constructing a suffix tree where N is the length of the input string, z is the number of seeds and D is the maximum edit-distance expected in the resulting repeats. The expected number of seeds is $E(z) = O(N^2 / \mid \Sigma \mid^{\lfloor L/D+1 \rfloor})$. This is suitable for finding maximal approximate repeats with small edit-distances. For long repeats and large edit-distances (that is, $L = c_1 N$ and $D = c_2 L$ where $c_1, c_2 < 1$), the expected running time of this algorithm is $O(N^5)$. This algorithm, therefore, is inefficient for extracting repeats for queries like "Find all maximal approximate repeats which are 10% different in edit-distance." Moreover, this algorithm would suffer from the poor locality behavior of the suffix tree [7], which results in many cache misses and dominates the running time.

Adebiyi, Jiang, and Kaufmann [1] designed algorithms for finding approximate non-tandem repeats by an idea similar to "seed expansion." The expected and practical running times of their algorithm are $O(DN^{1.7} \log N)$ and $O(N^{1.9} \log N)$, respectively, for DNA and protein sequences. However, it can only find short repeats with length $O(\log N)$. The main use of their algorithm is therefore for extracting short motifs only.

Instead of finding short approximate repeats of sequences as mentioned above, our paper finds the long conserved regions with relatively large edit-distance between multiple input sequences (genes). Moreover, our algorithm outputs only "significant" approximate repeats for diminishing the size of output such that the later analysis of the long conserved regions can be more efficient.

1.2 Definitions

Let S be a string of length $\mid S \mid = N$ over an alphabet Σ. $S[i]$ denotes the i-th character of S, for $i \in [1, N]$. For $i \leq j$, $S[i, j]$ denotes the substring of

S starting with the i-th and ending with the j-th character. Substring $S[i, j]$ can also be denoted by the pair of positions (i, j) sometimes for brevity and sometimes because the contents of the substrings are irrelevant. The length of the substring (i, j) is $\ell(i, j) = j - i + 1$.

There are three kinds of edit operations: deletions, insertions, and mismatches of single characters. The edit-distance of (i_1, j_1) and (i_2, j_2), denoted by $d((i_1, j_1), (i_2, j_2))$, is the minimum number of edit operations needed to transform $S[i_2, j_2]$ to $S[i_1, j_1]$.

A pair of positions (i_1, j_1), $i_1 \leq j_1$, covers a pair (i_2, j_2), $i_2 \leq j_2$, if and only if, $i_1 \leq i_2$ and $j_2 \leq j_1$. A pair $((i_1, j_1), (i_2, j_2))$ of pairs of string positions covers another pair $((i_3, j_3), (i_4, j_4))$ with respect to the first position if and only if (i_1, j_1) covers (i_3, j_3) and $\ell(i_1, j_1) \geq \ell(i_3, j_3) + 1$. A pair $((i_1, j_1), (i_2, j_2))$ of pairs of string positions completely covers another pair $((i_3, j_3), (i_4, j_4))$ if and only if (i_1, j_1) covers (i_3, j_3) and (i_2, j_2) covers (i_4, j_4). The length of $((i_1, j_1), (i_2, j_2))$ is defined to be $\ell((i_1, j_1), (i_2, j_2)) = \min\{j_1 - i_1 + 1, j_2 - i_2 + 1\}$.

A pair $((i_1, j_1), (i_2, j_2))$ of substrings is a (D, L)-approximate repeat if and only if $(i_1, j_1) \neq (i_2, j_2)$, $d((i_1, j_1), (i_2, j_2)) \leq D$ and $\ell((i_1, j_1), (i_2, j_2)) \geq L$. A (D, L)-approximate repeat is maximal if and only if it is not completely covered by any other (D, L)-approximate repeats. A (D, L)-maximal approximate repeat is supermaximal if and only if it is not covered by any other (D, L)-maximal approximate repeats with respect to the first position.

Two distinct (D, L)-maximal approximate repeats $((i_1, j_1), (i_2, j_2))$ and $((i_3, j_3), (i_4, j_4))$ are closely positioned if $i_1 = i_3$, $i_2 = i_4$. Note that, although $((i_1, j_1), (i_2, j_2))$ and $((i_3, j_3), (i_4, j_4))$ do not completely cover each other, (i_1, j_1) highly overlaps with (i_3, j_3), and (i_2, j_2) highly overlaps with (i_4, j_4). Figure 1 shows one such example. Worse, there are $O(D)$ such closely positioned (D, L)-maximal approximate repeats $((i_1, j_1), (i_2, j_2))$ given $i_1, i_2 \in [1, N]$ [6]. In the case that D is large, not all closely positioned (D, L)-maximal approximate repeats are interesting. Instead, if we only output (D, L)-supermaximal approximate repeats, then, given $i_1, i_2 \in [1, N]$, only one (D, L)-maximal approximate repeat $((i_1, j_1), (i_2, j_2))$ is output whose (i_1, j_1) is not covered by any other (i_3, j_3) for all (D, L)-maximal approximate repeats $((i_3, j_3), (i_4, j_4))$. This approach can efficiently diminish the size of output for the later analysis of the long conserved regions. Obviously, one useful application of extracting all (D, L)-supermaximal approximate repeats (β, β') of S is to find all longest possible substrings β of S such that there exists some other substring β' of S where $d(\beta, \beta') \leq D$ and $\ell(\beta, \beta') \geq L$.

This paper presents an algorithm that extracts all (D, L)-supermaximal approximate repeats of S given an input string S and thresholds L and D. This algorithm can be easily applied to the case where there are multiple input strings $S_1, S_2, \ldots, S_n$ by first concatenating the input strings into one long subject string S with a special symbol "$\sharp$" for separation: $S_1 \sharp S_2 \sharp \cdots \sharp S_n$.

The rest of this paper is organized as follows. Section 2 gives an efficient algorithm for a simplified problem. Section 3 gives an efficient algorithm for extracting all (D, L)-supermaximal approximate repeats based on the solution

Assume $D = 3$, $S[x, x+7]$= GAATCGGT and $S[y, y+6]$ = CTAGGCT. There are two (3,4)-maximal approximate repeats $((i_1, j_1), (i_2, j_2))$ and $((i_3, j_3), (i_4, j_4))$ where $i_1 = i_3 = x+1$, $i_2 = i_4 = y+1$, $j_1 = x+6$, $j_2 = y+4$, $j_3 = x+4$ and $j_4 = y+5$.

The first repeat $((i_1, j_1), (i_2, j_2))$:

$(i_1, j_1) = S[x+1, x+6]$=AATCGG
$(i_2, j_2) = S[y+1, y+4]$=TAGG

Alignment for $((i_1, j_1), (i_2, j_2))$:

```
(i1,j1)  GAATCGGT
(i2,j2)  CTA..GGC
```

The second repeat $((i_3, j_3), (i_4, j_4))$:

$(i_3, j_3) = S[x+1, x+4]$=AATC
$(i_4, j_4) = S[y+1, y+5]$=TAGGC

Alignment for $((i_3, j_3), (i_4, j_4))$:

```
(i3,j3)  G.AATCG
(i4,j4)  CTAGGCT
```

Note that, although $((i_1, j_1), (i_2, j_2))$ and $((i_3, j_3), (i_4, j_4))$ do not completely cover each other, (i_1, j_1) highly overlaps with (i_3, j_3), and (i_2, j_2) highly overlaps with (i_4, j_4). Outputting both of these maximal approximate repeats may not be interesting.

Fig. 1. An example of two closely positioned (3,4)-maximal approximate repeats

to the simplified problem. Section 4 proves the correctness of the algorithm and analyzes its time and space complexity. Moreover, Section 4 also shows that our algorithm is optimal in time when D is a constant.

2 The Simplified Problem

Before attacking the problem of finding all (D, L)-supermaximal approximate repeats, we first solve the problem of finding (D, L)-approximate repeats of the input string $S[1, N]$. Formally, we find, for each i, j, at least a pair of indices r and m such that $d((i, r), (j, m)) \leq D$, where $\ell(i, r) \geq L$ and $\ell(j, m) \geq L$ — if such r and m exist.

The idea is to find alignments with edit-distance k for every pair of S's substrings starting at i and j, respectively, where $1 \leq i, j \leq N$ and $1 \leq k \leq D$. More precisely, our algorithm computes $f_1(i,j,D)$ and $f_2(i,j,D)$ as the index $w+1$ of $S[i,w]$ and the index $h+1$ of $S[j,h]$, respectively, where $d((i,w),(j,h)) \le D$ for the maximum possible index w. The complete algorithm appears in Figure 2.

2.1 Analysis of FIND-SIM-REPEAT

Step 1 initializes the boundary values. For $S[i] \ne S[j]$, step 8 considers the possible edit-operations to align substrings starting from index i and j: insertion, deletion, or substitution. It is obvious that $d((i, f_1(i,j,D)-1),(j, f_2(i,j,D)-1)) \le D$ for $i > j$. It is also obvious that steps 9–18 assign $f_1(i,j,D)$ as the maximum value such that $d((i, f_1(i,j,D)-1),(j,m)) \le D$ for some m. If more than one of $f_1(i,j+1,k-1), f_1(i+1,j+1,k-1)$ and $f_1(i+1,j,k-1)$ are maximum, then FIND-SIM-REPEAT always chooses $f_1(x,y,k-1)$ such that $f_2(x,y,k-1)$ is the maximum of $f_2(i,j+1,k-1), f_2(i+1,j+1,k-1)$ and $f_2(i+1,j,k-1)$. Steps 21–26 prevent overlapping repeats to be output.

2.2 Time and Space Complexity

We analyze the time and space complexity. The running time is obviously $O(DN^2)$. Instead of using $O(DN^2)$ space as traditional dynamic programming does, we can reduce it to $O(DN)$. This is based on the observation that the only values needed to compute $f_1(i,j,k)$ are $f_1(i,j+1,k-1)$, $f_1(i+1,j+1,k-1)$, $f_1(i+1,j,k-1)$, and $f_1(i+1,j+1,k)$. Similarly, $f_2(i,j,k)$ depends only on $f_2(i,j+1,k-1)$, $f_2(i+1,j+1,k-1)$, $f_2(i+1,j,k-1)$, and $f_2(i+1,j+1,k)$. Note that when we compute the values $f_1(i,j,k)$, $j < i$ and $k = 1,\ldots,D$, those previously computed $f_1(i',j,k)$ with $i' > i+1$ are no longer needed. Only the "wavefront" is needed for the computation. This reduces the space complexity to $O(DN)$.

3 Extracting Supermaximal Approximate Repeats

The number of (D,L)-approximate repeats output by FIND-SIM-REPEAT can be very large, so this fact makes difficult the later analysis of long conserved regions. In order to extract only "significant" repeats, one idea is to output only (D,L)-maximal approximate repeats as in [6]. However, as discussed in Section 1.2, the number of (D,L)-maximal approximate repeats can still be large, especially when D is large. Therefore, we would like to output (D,L)-supermaximal approximate repeats only.

We present a two-phase procedure for extracting all (D,L)-supermaximal approximate repeats of the input string S. First, a modified version of FIND-SIM-REPEAT outputs substrings β for all (D,L)-supermaximal approximate repeats (β,β'). Second, according to those β output by the first phase, we apply another modified version of FIND-SIM-REPEAT to extract all (D,L)-supermaximal approximate repeats (β,β') and output them. We call the resulting algorithm FIND-SUPER-REPEAT.

```
 1: $f_1(N,j,k) = N, j = 1,\ldots,N, k = 0,\ldots,D$;
    $f_2(N,j,k) = j, j = 1,\ldots,N, k = 0,\ldots,D$;
    $f_1(j,j,k) = N, j = 1,\ldots,N, k = 0,\ldots,D$;
    $f_2(j,j,k) = N, j = 1,\ldots,N, k = 0,\ldots,D$;
 2: for $i = N-1,\ldots,1, j = i-1,\ldots,1$ do
 3:   if $S[i] = S[j]$ then
 4:     $f_1(i,j,k) = f_1(i+1,j+1,k), k = 0,\ldots,D$;
        $f_2(i,j,k) = f_1(i+1,j+1,k), k = 0,\ldots,D$;
 5:   else
 6:     $f_1(i,j,0) = i$;
        $f_2(i,j,0) = j$;
 7:     for $k = 1,\ldots,D$ do
 8:       Compare the values of $f_1(i,j+1,k-1), f_1(i+1,j+1,k-1)$ and $f_1(i+1,j,k-1)$;
 9:       if $f_1(i,j+1,k-1)$ is the maximum then
10:         $f_1(i,j,k) = f_1(i,j+1,k-1)$;
11:         $f_2(i,j,k) = f_2(i,j+1,k-1)$;
12:       else if $f_1(i+1,j+1,k-1)$ is the maximum then
13:         $f_1(i,j,k) = f_1(i+1,j+1,k-1)$;
14:         $f_2(i,j,k) = f_2(i+1,j+1,k-1)$;
15:       else
16:         $f_1(i,j,k) = f_1(i+1,j,k-1)$;
17:         $f_2(i,j,k) = f_2(i+1,j,k-1)$;
18:       end if
19:     end for
20:   end if
21:   if $f_2(i,j,D) \geq i$ then
22:     $f_2(i,j,D) = i$;
        $k = 1$;
23:     while $f_2(i,j,k) < i$ do
24:       $k++$;
25:     end while
        $f_1(i,j,D) = f_1(i,j,k-1) + (i - f_2(i,j,k-1))$;
26:   end if
27:   if $f_1(i,j,D) - i \geq L$ and $f_2(i,j,D) - j \geq L$ then
28:     Output $((i, f_1(i,j,D)-1), (j, f_2(i,j,D)-1))$
29:   end if
30: end for
```

Fig. 2. FIND-SIM-REPEAT

Lemma 1. *The set of (D,L)-approximate repeats of a string S output by FIND-SIM-REPEAT includes all (D,L)-supermaximal approximate repeats of S.*

Proof. Assume there is a (D,L)-supermaximal approximate repeat $((i,q),(j,m))$ which is not output by FIND-SIM-REPEAT. Then, this situation immediately results in a contradiction because, for the indices i and j, FIND-SIM-REPEAT always outputs the (D,L)-approximate repeat $((i,w),(j,h))$ such that $d((i,w),(j,h)) \leq D$ for the maximum possible index w. $\square$

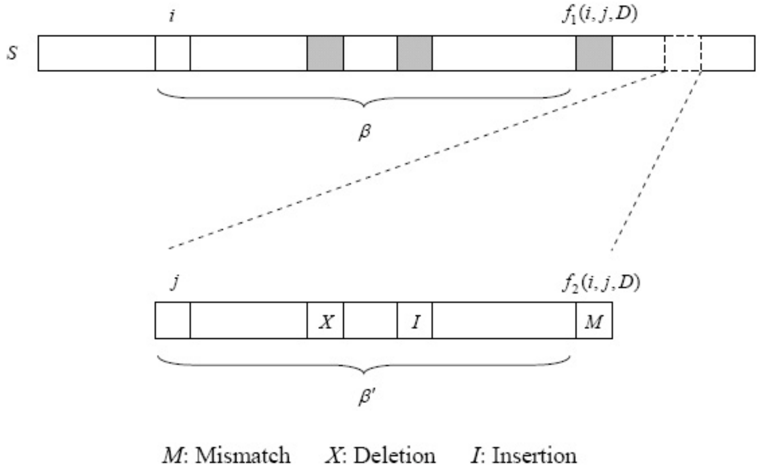

Fig. 3. An example of a $(2, L)$-approximate repeat (β, β') output by FIND-SIM-REPEAT where $D = 2$, $\beta = (i, f_1(i,j,D) - 1)$, $\beta' = (j, f_2(i,j,D) - 1)$ and the lengths of β and β' are $\ell(i, f_1(i,j,D) - 1) = f_1(i,j,D) - i \geq L$ and $\ell(j, f_2(i,j,D) - 1) = f_2(i,j,D) - j \geq L$, respectively.

Let U be the set of (D, L)-approximate repeats of S output by FIND-SIM-REPEAT. By Lemma 1, the first phase of FIND-SUPER-REPEAT finds β_1 for all (D, L)-supermaximal approximate repeats (β_1, β'_1) of S in U. Moreover, it only needs to focus on the substring β_2 of each (D, L)-approximate repeat (β_2, β'_2) in U for finding all of those β_1, so we can reduce the remaining work for the first phase of FIND-SUPER-REPEAT to maximal substrings problem defined below.

Finding Maximal Substrings. Given a string S and a set Q of substrings of S where $\mid S \mid= N$ and $\mid Q \mid= M$. We would like to find a subset W of Q such that none of the strings in W are covered by any other strings in Q. The time and space complexity are $O(M)$ and $O(N)$, respectively.

Let $P[1 \ldots N]$ be an array where $P[i] = -1$ for $i = 1, \ldots, N$ initially. The complete algorithm FIND-MAX-SUBSTRING appears in Figure 4. Recall that substring $S[x, y]$ is denoted by the pair of positions (x, y).

Lemma 2. *A substring (x, y) of S is output by FIND-MAX-SUBSTRING if and only if (x, y) is not covered by any other substrings of S in Q.*

Proof. If (x, y) is not covered by any other substrings of S in Q, then when (x, y) is picked up from Q in step 2, the "if condition" in step 3 must be satisfied. So, $P[x] = y$ before step 8 is executed. Moreover, $P[x]$ is always larger than maxTo in the x-th iteration of step 9. Otherwise, there must exist another substring (i, q) where $i < x$, $q \geq y$ and $P[x] = q$ such that (i, q) covers (x, y). This situation contradicts our assumption of (x, y). So, (x, y) is output.

```
 1: while $Q \neq \emptyset$ do
 2:    Pick a substring $(x, y)$ from $Q$ arbitrarily;
 3:    if $y > P[x]$ then
 4:       $P[x] = y$;
 5:    end if
 6: end while
 7: Let maxTo$= -1$;
 8: for $i = 1, \ldots, N$ do
 9:    if $P[i] >$ maxTo then
10:       Output $(i, P[i])$;
          maxTo$= P[i]$;
11:    end if
12: end for
```

Fig. 4. FIND-MAX-SUBSTRING

On the other hand, assume (x, y) is covered by some other substrings (i, q) of S in Q. If $i = x$ and $q > y$, then $P[x] = q$ before step 8 is executed and (x, y) will not be output. If $i < x$, $q \geq y$ and $P[x] = y$ before step 8 is executed, then the value of maxTo is at least q in the x-th iteration of step 9. Since maxTo $\geq q \geq y$, the "if condition" in step 9 will not be satisfied. (x, y) will not be output. □

Similar to FIND-MAX-SUBSTRING, in order to output β of all (D, L)-supermaximal approximate repeats (β, β'), we need to modify steps 27–29 and add some steps at the end of FIND-SIM-REPEAT. Figures 5 and 6 show this approach. The first phase of FIND-SUPER-REPEAT is this modified version of FIND-SIM-REPEAT. The functionality of REPEAT$[1 \ldots N]$ is similar to that of $P[1 \ldots N]$ in FIND-MAX-SUBSTRING and REPEAT$[i] = -1$ for $i = 1, \ldots, N$ initially.

```
if $f_1(i, j, D) - i \geq L$ and $f_2(i, j, D) - j \geq L$ and $f_1(i, j, D) - 1 >$ REPEAT$[i]$ then
   REPEAT$[i]= f_1(i, j, D) - 1$;
end if
```

Fig. 5. Modification of steps 27–29 of FIND-SIM-REPEAT for outputting β for all (D, L)-supermaximal approximate repeats (β, β')

```
maxTo $= -1$;
for $i = 1, \ldots, N$ do
   if REPEAT$[i] >$ maxTo then
      Output $(i,$REPEAT$[i])$;
      maxTo $=$ REPEAT$[i]$;
   end if
end for
```

Fig. 6. Additional steps at the end of FIND-SIM-REPEAT for outputting β for all (D, L)-supermaximal approximate repeats (β, β')

The second phase of FIND-SUPER-REPEAT is trivial. It simply follows FIND-SIM-REPEAT and outputs only (D, L)-supermaximal approximate repeats (β, β') according to those β output by the first phase.

4 Correctness and Analysis

4.1 Correctness

Theorem 1. *A (D, L)-approximate repeat of a string S is a (D, L)-supermaximal approximate repeat if and only if it is output by FIND-SUPER-REPEAT.*

Proof. Lemma 1 and 2 imply that the first phase of FIND-SUPER-REPEAT outputs β for all (D, L)-supermaximal approximate repeats (β, β'). Moreover, because Lemma 1 shows that the output of FIND-SIM-REPEAT must include all (D, L)-supermaximal approximate repeats of S, therefore, according to those β output by the first phase, the second phase of FIND-SUPER-REPEAT is able to extract all (D, L)-supermaximal approximate repeats (β, β'). As a result, a (D, L)-approximate repeat of S is a (D, L)-supermaximal approximate repeat if and only if it is output by FIND-SUPER-REPEAT. □

4.2 Time and Space Complexity

Recall that the time and space complexity of FIND-SIM-REPEAT are $O(DN^2)$ and $O(DN)$, respectively. The first phase of FIND-SUPER-REPEAT follows FIND-SIM-REPEAT with modifications as shown in Figures 5 and 6. Obviously, the steps of FIND-SUPER-REPEAT in Figure 5 do not increase the time complexity of FIND-SIM-REPEAT. Also, the time complexity of the steps of FIND-SUPER-REPEAT in Figure 6 is $O(N)$. On the other hand, the second phase of FIND-SUPER-REPEAT, in fact, is just to run FIND-SIM-REPEAT once. Therefore, the overall time complexity of FIND-SUPER-REPEAT is $O(DN^2)$.

Besides, the space complexity of FIND-SUPER-REPEAT is $O(DN)$. It is because FIND-SUPER-REPEAT only needs one more array REPEAT$[1 \ldots N]$ than FIND-SIM-REPEAT does and the space complexity of REPEAT$[1 \ldots N]$ is clearly $O(N)$.

4.3 Optimality of FIND-SUPER-REPEAT

Finally, we would like to show FIND-SUPER-REPEAT is optimal in time complexity when the input D is a constant. In addition, the space complexity of our algorithm is linear in N.

From Theorem 1, it is obvious that the time and space complexity of FIND-SUPER-REPEAT are $O(N^2)$ and $O(N)$, respectively, when D is a constant. Then, for each $e \in \Sigma^*$, we define the powers of e by $e^1 = e$, $e^2 = ee$, $e^3 = ee^2, \ldots, e^{n+1} = ee^n$ for any positive integer n. Let $\Sigma = \{A, T, G\}$, strings $S_1 = (A^{c_1}T^{c_2})^{N/2}$, and $S_2 = (G^{c_1}T^{c_2})^{N/2}$, where c_1, c_2 are positive constants. We concatenate S_1, S_2 into one string $S = S_1S_2$ to supply as the input of

FIND-SUPER-REPEAT. Then, given $D < c_1$, $L = c_2$, the number of all (D, L)-supermaximal approximate repeats of S is $\Omega(N^2)$. It is because there are $\Theta(N)$ of S's substrings $\beta = A^D T^{c_2}$ which can be aligned with $\Theta(N)$ of S's substrings $\beta' = G^D T^{c_2}$ such that the combinations of all (β, β') are (D, L)-supermaximal approximate repeats. As the time for outputting all (D, L)-supermaximal approximate repeats in this tight example is $\Omega(N^2)$, FIND-SUPER-REPEAT is optimal in time complexity when D is a constant.

References

1. E. F. Adebiyi, T. Jiang and M. Kaufmann (2001). "An Efficient Algorithm for Finding Short Approximate Non-tandem Repeats." *Bioinformatics*, 17(90001), pp. S5–S12.
2. G. Benson (1994). "A Space Efficient Algorithm for Finding the Best Nonoverlapping Alignment Score." In M. Crochemore and D. Gusfield, eds., *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM94)*, volume 807 of *LNCS*, pp. 1–14. Berlin: Springer Verlag.
3. W. Fitch, T. Smith and J. Breslow (1986). "Detecting Internally Repeated Sequences and Inferring the History of Duplication." In J. P. Segrest and J. J. Albers, eds., *Plasma Proteins. Part A: Preparation, Structure, and Molecular Biology*, volume 128 of *Methods in Enzymology*, pp. 773–788. San Diego, CA: Academic Press.
4. D. Gusfield (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* New York: Cambridge University Press.
5. S. K. Kannan and E. W. Myers (1996). "An Algorithm for Locating Nonoverlapping Regions of Maximum Alignment Score." *SIAM J. Computing*, 25(3), pp. 648–662.
6. S. Kurtz, E. Ohlebusch, et al. (2000). "Computation and Visualization of Degenerate Repeats in Complete Genomes." In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB2000)*, pp. 228–238.
7. S. Kurtz (1999). "Reducing the Space Requirement of Suffix Trees." *Software Practice and Experience*, 29(13), pp. 1149–1171.