MORE EFFICIENT ALGORITHMS FOR CLOSEST STRING AND SUBSTRING PROBLEMS [*]

BIN MA[†] AND XIAOMING SUN[‡]

Abstract.
The closest string problem and the closest substring problem are all natural theoretical computer science problems and find important applications in computational biology. Given n input strings, the closest string (substring) problem finds a new string within distance d to (a substring of) each input string and such that d is minimized. Both problems are NP-complete. In this paper we propose new algorithms for these two problems. For the closest string problem, we developed an exact algorithm with time complexity $O(n|\Sigma|^{O(d)})$, where Σ is the alphabet. This improves the previously best known result $O(nd^{O(d)})$, and results into a polynomial time algorithm when $d = O(\log n)$. By using this algorithm, a PTAS for closest string problem is also given with time complexity $O(n^{O(\epsilon^{-2})})$, improving the previously best known $O(n^{O(\epsilon^{-2}\log\frac{1}{\epsilon})})$ PTAS. A new algorithm for the closest substring problem is also proposed. Finally, we prove that a restricted version of the closest substring has the same parameterized complexity as closest substring, answering an open question in the literature.

1. Introduction.

The closest string and substring problems have been recently studied extensively in computational biology [16, 18, 22, 13, 24, 12, 23, 15, 7, 11, 29, 4, 26, 30]. The two problems have a variety of applications in bioinformatics, such as universal PCR primer design [20, 16, 5, 27, 12, 31], genetic probe design [16], antisense drug design [16, 4], finding unbiased consensus of a protein family [2], and motif finding [16, 12, 30, 3, 9]. In all these applications, a common task is to design a new DNA or protein sequence that is very similar to (a substring of) each of the given sequences. In the first three applications, the designed DNA sequence can bind to each of the given DNA sequences in order to perform its designated functions. In the last two applications, the designed sequence acts as an unbiased representative of all the given sequences. The common task has been formulated as the closest string problem and the closest substring problem.

Given n length-m strings s_1, s_2, …, s_n, and a radius d, the *closest string* problem seeks for a new length-m string s such that $d(s, s_i) \le d$. Here $d(\cdot,\cdot)$ is the Hamming distance. The *closest substring* problem seeks for a length L ($L < m$) string t such that for every $i = 1, 2, \dots, n$, there is a substring t_i of s_i with length L such that $d(t, t_i) \le d$. The problems may also be described as optimization problems where the objectives are to minimize the radius d.

Unfortunately, both of these two problems are NP-complete [10, 16]. In addition to many heuristic algorithms without any performance guarantee (for example [19, 23, 24]), researchers have developed approximation algorithms and fixed-parameter algorithms for the two problems. Approximation algorithms sacrifice the quality of the solution in order to achieve polynomial time [14]. A polynomial time approximation

[*]A preliminary version has appeared in the Proceedings of the 12th Annual International Conference on Research in Computational Molecular Biology (RECOMB'08).

[†]School of Computer Science, University of Waterloo, 200 University Ave. W, Waterloo, ON, Canada N2L3G1. Email: binma@uwaterloo.ca. Supported in part by China National High-tech R&D Program (863) 2008AA02Z313, China NSF 60553001, China National Basic Research Program 2007CB807900,2007CB807901, NSERC and Canada Research Chair.

[‡]Center for Advanced Study and Institute for Theoretical Computer Science, Tsinghua University, Beijing, China 100084. Email: xiaomings@tsinghua.edu.cn. Supported in part by China National NSF 60553001, 60603005, 60621062, and the National Basic Research Program 2007CB807900,2007CB807901.

scheme (PTAS) achieves ratio $1+\epsilon$ in polynomial time for any fixed $\epsilon > 0$. Fixed-parameter algorithms find optimal solutions with time complexity $f(k) \cdot n^c$ for a constant c and any function f [6]. Here k is a parameter naturally associated to the input instance.

For fixed-parameter algorithms, Stojanovic et. al [29] provided a linear time algorithm for $d = 1$. Gramm et. al [13] provided the first fixed-parameter algorithm for closest string with running time $O(nm + nd^{d+1})$. Therefore, for small values of d their algorithm can solve closest string in acceptable time. In this paper we present a novel algorithm that finds the optimal solution of closest string problem with running time $O\left(nm + nd \cdot (16|\Sigma|)^d\right)$. When the alphabet size is a constant and $d = O(\log n)$, this gives a polynomial time algorithm for the closest string problem.

The closest substring problem appeared to be harder than closest string in terms of parameterized complexity. For unbounded alphabet size, it has been shown that the problem is $W[1]$-hard even if all d, n, L are parameters [8, 9]. The $W[1]$-hardness indicates that the problem unlikely has a fixed-parameter polynomial time algorithm [6]. For $|\Sigma|$ being a constant or a parameter, the problem is $W[1]$-hard even if both d and n are parameters [22]. For a more complete review of the parameterized complexities of the closest substring problem, we refer the readers to [9, 22, 25]. Marx [22] gave a $|\Sigma|^{d(\log d+2)}(nm)^{\log d+O(1)}$ algorithm for the closest substring problem. In this paper we present a new algorithm for closest substring with improved time complexity $O\left((16|\Sigma|)^d \cdot nm^{\lceil \log d \rceil + 1}\right)$.

For approximation algorithms, Lanctot et. al [16] gave the first polynomial time approximation algorithm with approximation ratio $\frac{4}{3} + o(1)$. Li et. al [17] provided a PTAS for closest string with time complexity $O(mn^{O(\epsilon^{-5})})$. Ma [21] provided a PTAS for closest substring problem. These two PTAS results were collected in [18]. There have been many negative comments regarding the large exponent of the PTAS [9, 3, 11, 13, 22]. By using a lemma in [22] and an idea of [17], Andoni et. al [1] proposed a PTAS to obtain a much better time complexity $O(mn^{O(\epsilon^{-2}\log\frac{1}{\epsilon})})$. By combining our new fixed-parameter algorithm, in Section 5 we provide a simpler PTAS with further improved time complexity $O(mn^{O(\epsilon^{-2})})$.

Noticing the hardness of closest substring problem, Moan and Rusu [25] studied a more restricted version of closest substring. They put a diameter constraint on top of the original closest substring problem by further requiring the pairwise distances between substrings in the solution do not exceed a diameter D for some $D < 2d$. They hoped that such a constraint may reduce the parameterized complexity of closest substring when D is close enough to d. The condition for this to happen is left as a main open problem in [25]. In this paper we answer this question by proving that such condition does not exist. That is, for any given $\epsilon > 0$, all parameterized complexity results of closest string preserve in the constrained instances for $D < (1+\epsilon)d$.

2. Preliminaries and Notations. Let Σ be an alphabet with constant size $|\Sigma|$. Suppose s is a string over Σ. $|s|$ denotes the length of s. $s[i]$ denotes the i-th letter of s. Therefore, $s = s[1]s[2]\ldots s[m]$, where m is the length of s. Let s and t be two strings with the same length m, $d(s,t)$ denotes the Hamming distance between s and t. Use $[1,m]$ to denote the set $\{1, 2, \ldots, m\}$. For $P = \{i_1, i_2, \ldots, i_k\} \subseteq [1,m]$, define $s|_P = s[i_1]s[i_2]\ldots s[i_k]$ and $d^P(s,t) = d(s|_P, t|_P)$. Let $Q = [1,m] \setminus P$. From the definition of Hamming distance, clearly $d(s,t) = d^P(s,t) + d^Q(s,t)$. Let $Q(s,t)$ denote the set of positions where s and t agree, i.e., $Q(s,t) = \{j \mid s[j] = t[j]\}$. Similarly, for k given strings $s_1, s_2, \ldots, s_k$ of same length, $Q(s_1, s_2, \ldots, s_k)$ denotes the position set where all strings agree. Let $P(s,t)$ denote the position set where s and t disagree.

Let s_1, s_2, ..., s_n be n strings of length m. The *closest string problem* asks for a string center s such that $d = \max_{i=1}^{n} d(s, t_i)$ is minimized. The minimum value of d is called the *radius* of the n input strings. $D = \max_{1 \le i,j \le n} d(s_i, s_j)$ is called the *diameter* of the n input strings. Let $L \le m$. The *closest substring problem* asks for a length-L string center s and a length-L substring t_i of each s_i, such that $d = \max_{i=1}^{n} d(s, t_i)$ is minimized.

In this paper we will also study a more generalized version of closest string problem, the *neighbor string problem*: Given n strings s_1, s_2, ..., s_n with length m, and n nonnegative integers d_1, d_2, ..., d_n, the neighbor string problem seeks for a length m string s such that $d(s, s_i) \le d_i$ for every $1 \le i \le n$. An instance of the neighbor string problem is given by $\langle (s_1, d_1), (s_2, d_2), \dots, (s_n, d_n) \rangle$.

3. $O\left(nm + nd \cdot (16|\Sigma|)^d\right)$ Algorithm for Closest String Problem.

Parameterized complexity has been used to tackle NP-hard problems [6]. In principle, a fixed-parameter polynomial time algorithm is a well-structured superpolynomial algorithm such that the superpolynomial factor is only with respect to one parameter of the given instance. Many NP-hard problems have been found to be fixed-parameter tractable, which means that an algorithm with running time $f(k) \cdot n^c$ exists to solve the problem. Here k is a parameter naturally associated with the problem; n is the size of the input and c is a constant. Clearly $f(k)$ must be superpolynomial if $\mathbf{P} \neq \mathbf{NP}$. The hope is that this $f(k)$ will not grow too fast, and parameter k is small for practical instances; and hence the problem can be solved efficiently in practice.

Gramm et al. [13] provided a fixed-parameter polynomial time algorithm for closest string when the radius d is used as the fixed parameter. For a given instance $\{s_1, s_2, \dots, s_n\}$ and a given value d, their algorithm finds a center string s such that $d(s, s_i) \le d$ in $O(nm + nd^{d+1})$ time, if such a string exists.

In this section we provide a new algorithm for closest string problem with time complexity $O(nm + nd \cdot (16|\Sigma|)^d)$. When the alphabet size is a constant, our algorithm is exponentially faster than the previous algorithm. In order to design the algorithm for closest string, let us focus on the more generalized neighbor string problem.

LEMMA 3.1. [1] *Let $\langle (s_1, d_1), \dots, (s_n, d_n) \rangle$ be an instance of the neighbor string problem. If j satisfies $d(s_1, s_j) > d_j$, then for $Q = Q(s_1, s_j)$ and any solution s of the neighbor string problem, $d^Q(s, s_1) < \frac{d_1}{2}$.*

Proof. Let s be a solution, i.e. $d(s, s_i) \le d_i$ for $i = 1, 2, \dots, n$. Let $P = [1, m] \setminus Q$. Then each character in P contributes at least 1 to $d(s, s_1) + d(s, s_j)$, and each character in Q where s is not equal to s_1 (and s_j) contributes exactly 2. Since $d(s, s_1) + d(s, s_j) \le d_1 + d_j$ and $|P| > d_j$, the lemma follows. $\square$

Next we give an algorithm for the neighbor string problem in Figure 3.1. The idea of the algorithm is the following. Suppose there is s_{i_0} such that $d(s_1, s_{i_0}) > d_{i_0}$. Let $Q = Q(s_1, s_{i_0})$ be the positions where s_1 and s_{i_0} agree. $P = [1, |s_1|] \setminus Q$. Then we can "guess" the solution s on P, and construct a new neighbor string problem $\langle (s_1|_Q, e_1) \dots, (s_n|_Q, e_n) \rangle$. Because of Lemma 3.1, the new instance is such that $e_1 < \frac{d_1}{2}$. Thus, we can use the same algorithm to solve the new instance and the algorithm stops after at most $\lfloor \log d_1 \rfloor$ recursions. Because each recursion reduces d_1 by half, a careful analysis will show that the $\lfloor \log d_1 \rfloor$ recursions take $O\left(poly(nm) \times O(|\Sigma|)^{O(d)}\right)$ time for $d = \max d_i$.

THEOREM 3.2. *Let $d = \max_{1 \le i \le n} d_i$. If there is a solution s such that $d(s, s_i) \le$*

[1] Lemma 3.1 uses similar idea as Lemma 2.2 in [22]. However the lemma in [22] cannot be directly used in our algorithms.

```
Algorithm StringSearch
Input: An instance of neighbor string $\langle (s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n) \rangle$.
Output: String $s$ such that $d(s, s_i) \le d_i$ ($i = 1, \ldots, n$), or NULL if there
is no solution.
1. Try to find $i_0$ such that $d(s_1, s_{i_0}) > d_{i_0}$.
2. If step 1 fails, return $s_1$.
3. Let $Q = Q(s_1, s_{i_0})$, $P = [1, |s_1|] \setminus Q$.
4. For every possible string $t$ of length $|P|$ such that $d(t, s_1|_P) \le d_1$ and
   $d(t, s_{i_0}|_P) \le d_{i_0}$
4.1    Let $e_i = d_i - d(t, s_i|_P)$ for $i \ne 1$, and $e_1 = \min\{d_1 - d(t, s_1|_P), \lceil d_1/2 \rceil - 1\}$;
4.2    Use StringSearch to find the solution $u$ of
       $\langle (s_1|_Q, e_1), (s_2|_Q, e_2), \ldots, (s_n|_Q, e_n) \rangle$;
4.3    If $u \ne$ NULL then let $s|_P = t$ and $s|_Q = u$ and return $s$.
5. Return NULL.
```

FIG. 3.1. *The algorithm* **StringSearch**.

*d_i ($1 \le i \le n$), then algorithm **StringSearch** in Fig. 3.1 outputs a solution s' such that $d(s', s_i) \le d_i$ in time $O(mn + nd \cdot T(d, d_1))$, where the size of the search tree*

$$T(d, d_1) \le \binom{d + d_1}{d_1} (|\Sigma| - 1)^{d_1} \cdot 2^{2d_1}.$$

Proof. First let us prove the correctness of the algorithm. It is easy to verify that when the algorithm returns a non-null string in either line 2 or line 4.3, the string is a solution of the input instance. Let us prove that when there is a solution of the input instance, then the algorithm can find it. We prove this by using induction on d_1. If $d_1 = 0$ then clearly the algorithm is correct. When $d_1 > 0$ and line 1 finds i_0 successfully, by Lemma 3.1, the Q and P defined in line 3 are such that there is a solution s satisfying $d(s|_Q, s_1|_Q) \le e_1$. Therefore, this s is such that $d(s|_Q, s_i|_Q) \le e_i$ for $1 \le i \le n$. As a result, when $t = s|_P$ in line 4, by induction, the recursive call to Algorithm **StringSearch** at line 4.2 will find u such that $d(u, s_i|_Q) \le e_i$ for $1 \le i \le n$. Then it is easy to verify that the s returned in line 4.3 is a desired solution.

Next let us examine the time complexity of the algorithm **StringSearch**. We estimate the size (number of leaves) of the search tree first. In line 4, assume t is an eligible string and $d(t, s_1|_P) = k$. Then $|P| = d(s_1|_P, s_{i_0}|_P) \le d(t|_P, s_1|_P) + d(t|_P, s_{i_0}|_P) \le d_{i_0} + k \le d + k$. Therefore, there are at most $\binom{|P|}{k}(|\Sigma| - 1)^k \le \binom{d+k}{k}(|\Sigma| - 1)^k$ such strings t. For each of them, the size of the subtree rooted at t of the search tree is bounded by $T(d, \min\{d_1 - k, \lceil d_1/2 \rceil - 1\})$. k can take values from 0 to d_1. Therefore, the search tree size satisfies

$$\begin{aligned} T(d, d_1) \le & \sum_{k=\lfloor d_1/2 \rfloor + 1}^{d_1} \binom{d+k}{k} (|\Sigma| - 1)^k T(d, d_1 - k) \\ & + \sum_{k=0}^{\lfloor d_1/2 \rfloor} \binom{d+k}{k} (|\Sigma| - 1)^k T(d, \lceil d_1/2 \rceil - 1) \qquad (3.1) \end{aligned}$$

Clearly $T(d, 0) = 1$ because in this case s_1 is the solution. We prove by induction that for $\tilde{d} \geq 1$,

$$T(d, \tilde{d}) \leq 2^{2\tilde{d}} \binom{d+\tilde{d}}{\tilde{d}} (|\Sigma|-1)^{\tilde{d}}. \tag{3.2}$$

It is easy to verify that when $\tilde{d} = 1$, $T(d, 1) \leq (d+1)(|\Sigma|-1)+1$, the statement is true. When $\tilde{d} = 2$, because of (3.1), $T(d, 2) \leq \binom{d+2}{2}(|\Sigma|-1)^2 + (d+1)(|\Sigma|-1)+1 \leq 2\binom{d+2}{2}(|\Sigma|-1)^2$, the statement is also true. Next we suppose $d_1 > 2$ and eq. (3.2) is true for $0 \leq \tilde{d} < d_1$. We bound the two terms of (3.1) separately. Let $k_0 = \lfloor d_1/2 \rfloor + 1$. The first term is

$$\begin{aligned}
& \sum_{k=k_0}^{d_1} \binom{d+k}{k} (|\Sigma|-1)^k T(d, d_1-k) \\
\leq & \sum_{k=k_0}^{d_1} \binom{d+d_1}{k} (|\Sigma|-1)^k T(d, d_1-k) \\
\leq & \sum_{k=k_0}^{d_1} \binom{d+d_1}{k} (|\Sigma|-1)^k \cdot \binom{d+d_1-k}{d_1-k} (|\Sigma|-1)^{d_1-k} \cdot 2^{2(d_1-k)} \\
= & \binom{d+d_1}{d_1} (|\Sigma|-1)^{d_1} \sum_{k=k_0}^{d_1} \binom{d_1}{k} \cdot 2^{2(d_1-k)} \qquad (3.3) \\
\leq & \binom{d+d_1}{d_1} (|\Sigma|-1)^{d_1} \cdot 2^{d_1-1} \sum_{k=k_0}^{d_1} \binom{d_1}{k} \\
\leq & \binom{d+d_1}{d_1} (|\Sigma|-1)^{d_1} \cdot 2^{2d_1-2}. \qquad (3.4)
\end{aligned}$$

Here the equality (3.3) is because $\binom{d+d_1}{k} \times \binom{d+d_1-k}{d_1-k} = \frac{(d+d_1)!}{k!(d+d_1-k)!} \times \frac{(d+d_1-k)!}{(d_1-k)!d!} = \frac{(d+d_1)!}{d_1!d!} \times \frac{d_1!}{(d_1-k)!k!} = \binom{d+d_1}{d_1} \times \binom{d_1}{k}$. The rest of the proof is to bound the second term by $3\binom{d+d_1}{d_1}(|\Sigma|-1)^{d_1} \cdot 2^{2d_1-2}$. The second term is

$$\begin{aligned}
& \sum_{k=0}^{k_0-1} \binom{d+k}{k} (|\Sigma|-1)^{k_0} T(d, d_1-k_0) \\
\leq & \sum_{k=0}^{k_0-1} \binom{d+k}{k} (|\Sigma|-1)^{k_0} \cdot \binom{d+d_1-k_0}{d_1-k_0} (|\Sigma|-1)^{d_1-k_0} \cdot 2^{2(d_1-k_0)} \\
= & \binom{d+d_1-k_0}{d_1-k_0} (|\Sigma|-1)^{d_1} \cdot 2^{2(d_1-k_0)} \sum_{k=0}^{k_0-1} \binom{d+k}{k} \\
\leq & \binom{d+d_1-k_0}{d_1-k_0} (|\Sigma|-1)^{d_1} \cdot 2^{2(d_1-k_0)} \binom{d+k_0}{k_0}.
\end{aligned}$$

So we only need to prove

$$\binom{d+d_1-k_0}{d_1-k_0} \binom{d+k_0}{k_0} 2^{-2k_0} \leq \frac{3}{4} \cdot \binom{d+d_1}{d_1},$$

or equivalently,

$$\binom{d+d_1-k_0}{d_1-k_0}\binom{d_1}{k_0} \le \frac{3}{4}\cdot 2^{2k_0}\binom{d+d_1}{d_1-k_0}. \tag{3.5}$$

(3.5) is true because

$$\binom{d+d_1-k_0}{d_1-k_0} \le \binom{d+d_1}{d_1-k_0}, \quad \binom{d_1}{k_0} \le \frac{1}{2}\cdot 2^{d_1+1} < \frac{3}{4}\cdot 2^{2k_0}.$$

Hence (3.2) is correct.

At each leaf, the time complexity of line 1 is $O(nm)$. By carefully remembering the previous distances and only update the $O(d)$ positions changed, this can be done in $O(nd)$ time. The total running time is dominated by the leaves. Therefore, the time complexity of the algorithm is $O(nm + nd \cdot T(d, d_1))$. □

Corollary 3.3. ***StringSearch*** *solves the closest string problem in time*

$$O\left(nm + nd \cdot 2^{4d}(|\Sigma|-1)^d\right).$$

Remark: In line 4 of the algorithm StringSearch, one cannot only enumerate the string t of which each letter is taken from either s_1 or s_{i_0}. This is because when $d(s_1, s_{i_0}) < d_1 + d_{i_0}$, the solution s may be different from both s_1 and s_{i_0}. For this reason, all letters in Σ need to be tried, resulting in a time complexity related to $|\Sigma|$.

4. More Efficient Algorithm For Closest Substring.

In [22], an algorithm with running time $|\Sigma|^{d(\log_2 d+2)}N^{\log_2 d+O(1)}$ is given, where N is the total length of the input strings. In this section we improve it to $O\left(n|\Sigma|^{O(d)}m^{\lceil\log_2 d\rceil+1}\right)$. That is, the $\log_2 d$ factor at the exponent of $|\Sigma|^{d(\log_2 d+2)}$ is removed. Moreover, the total length N is replaced by the length m of the longest input string.

Again, in order to develop an algorithm for closest substring, we attempt to solve a more generalized version of closest substring. For convenience, we call the new problem *partial knowledge closest substring.* An instance of the partial knowledge closest substring problem is given by $\langle\{s_1, s_2, \ldots, s_n\}, d, d_1, L, O, \tilde{t}\rangle$, where $O \subset [1, L]$ and $\tilde{t}$ is a string of length $|O|$. The problem is to find a string t of length L, such that $t|_O = \tilde{t}$, $d^{[1,L]\setminus O}(t, s_1) \le d_1$, and for every i, $d(t, t_i) \le d$ for at least one substring t_i of s_i.

Theorem 4.1. *Algorithm* ***SubstringTry*** *in Fig. 4.1 finds a solution for closest substring with time complexity*

$$O\left(nL + nd \cdot 2^{4d}|\Sigma|^d \cdot m^{\lceil\log_2 d\rceil+1}\right).$$

Proof. (*Sketch*) When all the input strings have the same length L, a careful comparison between Algorithm **SubstringSearch** and the previous Algorithm **StringSearch** can see that the two algorithms are equivalent. The only difference is made when $|s_i| > L$. Then the "guess" operation in line 4 requires the algorithm to try all possible substrings of s_{i_0}. This expands the search tree size by a factor of at most m. Because of Lemma 3.1, the recursion of Algorithm **SubstringSearch** is at most $\lceil\log_2 d\rceil$ levels. This increases the search tree size by a factor of $m^{\lceil\log_2 d\rceil}$. Combining with Corollary 3.3, the theorem is proved. □

```
Algorithm SubstringSearch
Input: $\langle \{s_1, s_2, \ldots, s_n\}, d, L, O, \tilde{t} \rangle$ such that $|s_1| = L$.
Output: A solution $t$ of the partial knowledge closest substring, or NULL
if there is no solution.
1. Let $O' = [1..L] \setminus O$. Let $s$ be a string such that $s|_O = \tilde{t}$ and $s|_{O'} = s_1|_{O'}$.
2. Try to find $i_0$ such that $d(s, t_{i_0}) > d_{i_0}$ for every substring $t_{i_0}$ of $s_{i_0}$.
3. If line 1 fails, return $s$.
4. Guess a substring $t_{i_0}$ of $s_{i_0}$.
5. Let $P = P(s_1, t_{i_0}) \setminus O$.
6. For every possible string $t$ of length $|P|$ such that $d(t, s_1|_P) \leq d_1$ and
   $d(t, t_{i_0}|_P) \leq d - d(\tilde{t}, t_{i_0}|_O)$
6.1     Let $t'$ be a string such that $t'|_O = \tilde{t}$ and $t'|_P = t$.
6.2     Let $e_1 = \min\{d_1 - d(t, s_1|_P), \lceil d_1/2 \rceil - 1\}$.
6.3     Use SubstringSearch to find solution $u$ of
        $\langle \{s_1, s_2, \ldots, s_n\}, d, e_1, L, O \cup P, t'|_{O \cup P} \rangle$.
6.4     If 6.3 is successful then return $u$.
7. Return NULL.
```

```
Algorithm SubstringTry
Input: $\langle \{s_1, s_2, \ldots, s_n\}, d, L \rangle$.
1. for every length $L$ substring $t_1$ of $s_1$,
1.1   call SubstringSearch with $\langle \{t_1, s_2, \ldots, s_n\}, d, d, L, \emptyset, e \rangle$.
```

FIG. 4.1. *The algorithms* **SubstringSearch** *and* **SubstringTry**.

5. More Efficient PTAS for Closest String. In [17, 18], a PTAS for closest string problem was given. To achieve approximation ratio $1 + \epsilon$, the running time of the algorithm was $O\left(mn^{O(\epsilon^{-5})}\right)$. Apparently this PTAS has only theoretical value as the degree of the polynomial grows very fast when ϵ gets small. By using the Lemma 2.2 in [22] and an idea of [17, 18], Andoni et. al [1] proposed a PTAS in [17] to get much better time complexity $O(mn^{O(\epsilon^{-2} \log \frac{1}{\epsilon})})$. The proof in [1] argued that when $d = \Omega(\log n/\epsilon^2)$, a standard linear programming relaxation method can solve the closest string problem with good approximation ratio. When $d = O(\log n/\epsilon^2)$, one can exhaustively enumerate all the possibilities of positions in the solution where r of the input strings do not agree. However, by using Lemma 2.2 of [22], r can be reduced from the original $O(\frac{1}{\epsilon})$ in [18] to $O(\log \frac{1}{\epsilon})$.

With our new fixed-parameter algorithm that runs $O\left(mn + nd \cdot (16|\Sigma|^d)\right)$ time, we can further reduce the time complexity by the following algorithm: Use the fixed-parameter algorithm to solve $d = O(\log n/\epsilon^2)$, and use the standard linear programming relaxation to solve the case $d = \Omega(\log n/\epsilon^2)$. It is easy to verify that this provides a simple $O(m \cdot n^{O(\epsilon^{-2})})$ PTAS.

THEOREM 5.1. *Closest string has a PTAS that achieves approximation ratio* $1+\epsilon$ *with time* $O(m \cdot n^{O(\epsilon^{-2})})$.

6. Hardness result. Together with the development of fixed-parameter polynomial time algorithms, W-hierarchy has been developed to prove fixed-parameter intractability [6]. The W[1]-hardness results reviewed in Section 1 indicate that the closest substring problem unlikely has fixed-parameter polynomial time algorithms even if both d and n are fixed-parameters. More parameterized complexity results about the closest substring problem can be found in [9, 22, 25].

Moan and Rusu [25] studied a variant of the closest substring problem by adding a constraint on the diameter of the solution, and hoped that the constraint can help reduce the parameterized complexity of the problem. The constraint is called the bounded Hamming distance (BHD) constraint in their paper. Then the *BHD-constrained closest substring (BCCS)* problem is defined as follows.

BCCS Given a set of n strings $s_1, s_2, \ldots, s_n$, substring length L, radius d, and diameter D. Find length-L substring t_i of each s_i, $i = 1, 2, \ldots, n$, and a new length-L string t, such that $d(t_i, t_j) \leq D$, and $d(t, t_i) \leq d$.

Clearly, $d \leq D \leq 2d$. For any $c \geq \frac{4}{3}$, Moan and Rusu proved that the diameter constraint $D \leq c \cdot d$ does not reduce the complexity of closest substring problem. More precisely, with any $c \geq \frac{4}{3}$, all parameterized complexity results for closest substring preserve for BCCS when using any non-empty subset of the following values as parameters: the radius d, the alphabet size $|\Sigma|$, the number of input strings n, the length of desired substrings L.

However, Moan and Rusu pointed out that in computational biology, D is usually significantly smaller than $2d$. Therefore, they hoped that when $\frac{D}{d}$ is very close to 1, the BCCS problem might become easier than the original closest substring problem. If this is true, BCCS can be used to solve the practical closest substring problems. The finding of the necessary condition for that BCCS problem becomes easier is left as the "main open question" of the paper [25]. In this section, we answer this question negatively with the following theorem.

THEOREM 6.1. *For any constant $\epsilon > 0$, with the diameter constraint $D \leq (1+\epsilon)d$, all parameterized complexity ($W[l]$-hardness) results for closest substring preserve for BCCS when using any non-empty subset of the following values as parameters: the radius d, the alphabet size $|\Sigma|$, the number of input strings n, the length of desired substrings L.*

Proof. The proof is done in three steps: First, we construct an instance of closest string with radius $\tilde{d}$ and diameter $\tilde{D} = (1 + o(1))\tilde{d}$. Then, we show that an instance of closest substring with radius d and diameter D can be "merged" with an instance of the closest string with radius $\tilde{d}$ and diameter $\tilde{D}$, so that the new instance has radius $d + \tilde{d}$ and diameter $D + \tilde{D}$. Thirdly, by letting $\tilde{d} \gg d$ and $\tilde{D} \gg D$, we get an instance such that the diameter is very close to the radius. Thus, we reduce the closest substring problem to BCCS, and hence prove the theorem. The details are given in the following.

STEP I

First let us construct an instance $\mathcal{I}_1$ of the closest string problem with very close radius and diameter. Let k be an even number. Examine the instance with k binary strings $x_1, x_2, \ldots, x_k$. Each x_i has length $\tilde{L} = \binom{k}{k/2}$. For each column j, exactly half of $x_1[j], x_2[j], \ldots, x_k[j]$ take value 0 and the other half take value 1. Hence there are in total $\binom{k}{k/2}$ ways to assign values to a column. Each of the $\binom{k}{k/2}$ columns takes a distinct way.

Claim 1 The radius of the constructed instance is $\tilde{d} = \tilde{L}/2$.

Proof. Because of the construction, each string has half of the $\tilde{L}$ letters as 0. Therefore, $d(0^{\tilde{L}}, x_i) = \tilde{L}/2$ for every x_i. Therefore, the radius is at most $\tilde{L}/2$.

On the other hand, for any center string x, at each column, the total number of differences between x_i $(i = 1, 2, \ldots, k)$ and the center string is exactly $k/2$. Therefore, $\sum_{i=1}^{k} d(x, x_i) = k\tilde{L}/2$. Consequently, $\max_{i=1}^{k} d(x, x_i) \geq \tilde{L}/2$. The claim is proved. □

Now let us examine the diameter of the constructed instance. For every two strings x_i and x_j, the Hamming distance is the number of columns such that x_i and

x_j take different values. This is equivalent to the number of ways to split k elements into two equal-sized subsets, ensuring that elements i and j are separated. With simple combinatorics, this number is $2\binom{k-2}{\frac{k}{2}-1}$. Therefore,

$$\frac{\tilde{D}}{\tilde{d}} = \frac{2\binom{k-2}{\frac{k}{2}-1}}{\tilde{L}/2} = \frac{4\binom{k-2}{\frac{k}{2}-1}}{\binom{k}{k/2}} = \frac{k}{k-1}$$

In order to avoid the exponential growth of $\tilde{D}$ and $\tilde{d}$ with respect to k, we note that $\tilde{D}$ and $\tilde{d}$ can be enlarged while keeping the same ratio $\frac{\tilde{D}}{\tilde{d}}$ by replacing each x_i by $\underbrace{x_i x_i \dots x_i}_{K}$, i.e., the concatenation of K copies of x_i. In the rest of the proof we consider $\mathcal{I}_1$ as such an enlarged instance, and the value K is to be determined later. The notations diameter $\tilde{D}$, radius $\tilde{d}$, input string x_i, and string length $\tilde{L}$ all correspond to the enlarged instance.

Step II

Let $\mathcal{I} = \langle \{s_1, \dots, s_n\}, L, d \rangle$ be an instance of the closest substring. We construct a new instance in the following.

For each s_i $(i = 1, \dots, n)$ and each x_j $(j = 1, \dots, k)$, let

$$s_{ij} = X\ s_i[1..L]x_j\ XY_{ij}X\ s_i[2..L+1]x_j\ XY_{ij}X\ \dots\ XY_{ij}X\ s_i[m-L+1..m]x_j\ X.$$

Here X and Y_ij are binary strings (to-be-constructed) that serve as separators. The new instance is then

$$\mathcal{I}_2 = \langle \{s_{ij} | i = 1, \dots, n, j = 1, \dots, k\}, 2|X| + L + \tilde{L}, d + \tilde{d} \rangle.$$

The X separators will ensure that they are aligned together in a solution, and the Y_{ij} separators will ensure that a solution does not include two adjacent $s_i[k..L+k-1]$ segments. X and Y_{ij} are designed as follows.

For $N > 0$, let $K = \{kN^2 + k(k+1)/2 \,|\, k = 1, 2, \dots, N-1\}$. Then K is a Golomb ruler, where the difference between every pair of numbers in K is distinct. Let X be a length-N^3 binary string on alphabet $\{*, \#\}$ such that $X[k] = *$ if and only if $k \in K$. Denote $X[k + N^3] = X[k]$. Then for every k, we have

$$d(X, X[k..N^3 + k - 1]) \geq N - 2. \qquad (6.1)$$

Therefore, for sufficiently large N, a solution must align X exactly together.

Similarly to K, we can design nk sets $K_{ij} \subset [1, M]$, $(i = 1, \dots, n, j = 1, \dots, k)$, such that (i) $|K_{ij}| = N'$ and (ii) $|k_1 - k_2| = |k'_1 - k'_2|$ for $\{k_1, k_2\} \subset K_{ij}$ and $\{k'_1, k'_2\} \subset K_{i'j'} \Rightarrow \{k_1, k_2\} = \{k'_1, k'_2\}$ and $i = i'$ and $j = j'$. We leave the proof to the readers. Let $Y_{i,j}$ be a length-M binary 0-1 string such that $Y_{i,j}[k] = 1$ if and only if $k \in K_{ij}$. Denote $Y_{ij}[k + M] = Y_{ij}[k]$. Then for any k, we have

$$d(Y_{ij}, Y_{i'j'}[k..M + k - 1]) \geq N' - 1. \qquad (6.2)$$

Thus, for sufficiently large N', a solution cannot include the Y_{ij} segment.

Claim 2 $\mathcal{I}$ has a solution with radius $\leq d$ and diameter $\leq D$ if and only if $\mathcal{I}_2$ has a solution with radius $\leq d + \tilde{d}$ and diameter $\leq D + \tilde{D}$.

Proof. Suppose $\mathcal{I}$ has a solution $s_i[l_i..l_i + L - 1]$, $i = 1, \dots, n$ with radius d and diameter D. Then the substrings $Xs_i[l_i..l_i + L - 1]x_jX$, $i = 1, \dots, n$, $j = 1, \dots, k$

are a solution of $\mathcal{I}_2$. It is easy to verify that the radius and diameter are bounded by $d+\tilde{d}$ and $D+\tilde{D}$, respectively.

Now we prove the other direction. From the construction of X and Y_{ij}, we know that the solution of $\mathcal{I}_2$ contains only one $s_i[k..L+k-1]$ from each s_{ij}, and is such that X from different strings are aligned exactly together. Further, without making the solution worse, we can easily modify the solution by "sliding" so that every substring has the form $Xs_i[l_i..l_i+L-1]x_jX$ for some l_i. Next we show that $s_i[l_i..l_i+L-1]$ $(i=1,2,\ldots,n)$ is the desired solution for $\mathcal{I}$.

Let $Xs\tilde{s}X$ be the center of $Xs_i[l_i..l_i+L-1]x_jX$ with radius $d+\tilde{d}$. Because $\tilde{d}$ is the radius of $\mathcal{I}_1$, there is j_0 such that $d(\tilde{s},x_{j_0})=\tilde{d}$. Therefore, $d(Xs\tilde{s}X, Xs_i[l_i..l_i+L-1]x_{j_0}X)\le d+\tilde{d}$ derives that $d(s,s_i[l_i..l_i+L-1])\le d$ for every i.

Similarly, there are j_0 and j_1 such that $d(x_{j_0},x_{j_1})=\tilde{D}$. Therefore, $d(Xs_i[l_i..l_i+L-1]x_{j_0}X, Xs_{i'}[l_{i'}..l_{i'}+L-1]x_{j_0}X)\le D+\tilde{D}$ derives that $d(s_i[l_i..l_i+L-1], s_{i'}[l_{i'}..l_{i'}+L-1])\le D$ for every i and i'.

The claim is proved. □

Step III

For any $\epsilon>0$, we let $k=\lceil \frac{2}{\epsilon}+1\rceil$ and $K=\left\lceil \frac{4D}{\binom{k}{k/2}\epsilon}\right\rceil$ in the construction of $\mathcal{I}_1$. Then $\tilde{d}=\frac{K}{2}\cdot\binom{k}{k/2}\ge\frac{2D}{\epsilon}$. Then in instance $\mathcal{I}_2$, the ratio between the diameter and radius is

$$\frac{D+\tilde{D}}{d+\tilde{d}}\le\frac{D}{\tilde{d}}+\frac{\tilde{D}}{\tilde{d}}\le\frac{\epsilon}{2}+\frac{k}{k-1}\le 1+\epsilon$$

Thus, we successfully reduce the closest substring problem to the closest substring problem with the constraint that the diameter is within $1+\epsilon$ times the radius. The number n, length m of the input strings are increased only by a constant factor determined by ϵ. The new length L and radius d of the substrings are polynomials of the old L and d. Therefore, all the W-complexities of the closest substring problem still hold with the diameter constraint $D\le(1+\epsilon)d$ for any small $\epsilon>0$. The theorem is proved. □

7. Discussion. The closest string and closest substring are two problems motivated and well-studied in computational biology. We proposed a novel technique that leads to more efficient fixed-parameter algorithm for closest string. This is also the first polynomial algorithm for the problem when $d=O(\log n)$. The same technique is then used to give a more efficient algorithm for closest substring. As a consequence of the fixed-parameter algorithm, we presented a more efficient PTAS of the closest string problem. At last, we showed that a restricted version of the closest substring problem has the same parameterized complexity as the original closest substring problem. This answers an open question raised in [25].

An interesting observation is that the approximation and fixed-parameter strategies work complementarily for different d values. For smaller $d<\log_2 n$ and binary strings, our fixed-parameter algorithm has time complexity $O(nm+nd\cdot 2^{4d})=O(nm+n^5\log_2 n)$. For larger $d>c\ln n/\epsilon^2$ for some constant c, the linear programming relaxation's time complexity is dominated by the time to solve a linear programming of m variables and nm coefficients, which is again a low-degree polynomial. This scenario can be intuitively explained as follows. When d is small and n is large, each input string puts a strong constraint on the solution, and consequently removes a large portion of the search space in a fixed-parameter algorithm. Therefore, it is easier to design a fixed-parameter algorithm. Conversely, when d is large and n is small, the constraint superimposed by each input string is weaker and there are fewer constraints. Therefore, it is easier to find an approximate solution to roughly satisfy those constraints.

But when d falls in between $\log_2 n$ and $c \ln n/\epsilon^2$, the polynomial will have high degree for the fixed-parameter algorithm, and the approximation ratio of the linear programming relaxation may exceed $1+\epsilon$. The instances with d in this range seem to be the "hardest" instances of the closest string problem. However, because the fixed-parameter algorithm has polynomial (although with high degree) running time on these instances, a proof for the "hardness" of these instances seems to be difficult too. We leave the finding of a more efficient (approximation) algorithm for $\log_2 n < d < c \ln n/\epsilon^2$ as an open problem. Another open problem is the finding of $O(poly(nm)c^d)$ time algorithm for unbounded alphabet size and a constant c.

Acknowledgement. Bin Ma's work was partially done when he visited Professor Andrew Yao in ITCS at Tsinghua University in 2006 and 2007, and when he was an associate professor at the University of Western Ontario. The authors thank an anonymous referee for suggesting a more intuitive presentation of the proof of Lemma 3.1.

REFERENCES

[1] A. Andoni, P. Indyk, and M. Patrascu. On the optimality of the dimensionality reduction method. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 449–458, 2006.

[2] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi. Banishing bias from consensus sequences. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes In Computer Science(1264)*, pages 247 – 261, 1997.

[3] J. Davila, S. Balla, and S. Rajasekaran. Space and time efficient algorithms for planted motif search. In *International Conference on Computational Science (2)*, pages 822–829, 2006.

[4] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang. Genetic design of drugs without side-effects. *SIAM Journal on Computing 32(4)*, pages 1073–1090, 2003.

[5] J. Dopazo, A. Rodríguez, J.C. Sáiz, and F. Sobrino. Design of primers for PCR amplification of highly variable genomes. *CABIOS*, 9:123–125, 1993.

[6] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

[7] P. A. Evans and A. D. Smith. Complexity of approximating closest substring problems. In *FCT*, pages 210–221, 2003.

[8] P.A. Evans, A.D. Smith, and H.T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-3):407–430, 2003.

[9] M.R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26(2):141–167, 2006.

[10] M. Frances and A. Litman. On covering problems of codes. *Theoretical Computer Science*, 30:113–119, 1997.

[11] J. Gramm, J. Guo, and R. Niedermeier. On exact and approximation algorithms for distinguishing substring selection. In *FCT*, pages 159–209, 2003.

[12] J. Gramm, F. Hüffner, and R. Niedermeier. Closest strings, primer design, and motif search. In *L. Florea et al. (eds), Currents in Computational Molecular Biology, poster abstracts of RECOMB 2002*, pages 74–75, 2002.

[13] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37:25–42, 2003.

[14] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, 1996.

[15] Y. Jiao, J. Xu, and M. Li. On the k-closest substring and k-consensus pattern problems. In *Combinatorial Pattern Matching: 15th Annual Symposium (CPM 2004), 3109 volume of Lecture Notes in Computer Science*, pages 130–144, 2004.

[16] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string search problems. In

Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 633–642, 1999.

[17] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 473–482, 1999.

[18] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.

[19] X. Liu, H. He, and O. Sýkora. Parallel genetic algorithm and parallel simulated annealing algorithm for the closest string problem. In *Advanced Data Mining and Applications, LNCS 3584*, pages 591–597, 2005.

[20] K. Lucas, M. Busch, S. MÖssinger, and J.A. Thompson. An improved microcomputer program for finding gene- or gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. *CABIOS*, 7:525–529, 1991.

[21] B. Ma. A polynomial time approximation scheme for the closest substring problem. In *Proceedings of the 11th Symposium on Combinatorial Pattern Matching*, pages 99–107, 2000.

[22] D. Marx. The closest substring problem with small distances. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 63–72, 2005.

[23] H. Mauch, M. J. Melzer, and J. S. Hu. Genetic algorithm approach for the closest string problem. In *Proceedings of the 2nd IEEE Computer Society Bioinformatics Conference (CSB)*, pages 560–561, 2003.

[24] C. N. Meneses, Z. Lu, C. A. S. Oliveira, and P. M. Pardalos. Optimal solutions for the closest-string problem via integer programming. *INFORMS Journal on Computing*, 2004.

[25] C. Moan and I. Rusu. Hard problems in similarity searching. *Discrete Applied Mathematics*, 144:213–227, 2004.

[26] F. Nicolas and E. Rivals. Complexities of the centre and median string problems. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, pages 315–327, 2003.

[27] V. Proutski and E.C. Holme. Primer master: A new program for the design and analysis of PCR primers. *CABIOS*, 12:253–255, 1996.

[28] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer program. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[29] N. Stojanovic, P. Berman, D. Gumucio, R. Hardison, and W. Miller. A linear-time algorithm for the 1-mismatch problem. In *Proceedings of the 5th International Workshop on Algorithms and Data Structures*, pages 126–135, 1997.

[30] L. Wang and L. Dong. Randomized algorithms for motif detection. *Journal of Bioinformatics and Computational Biology*, 3(5):1039–1052, 2005.

[31] Y. Wang, W. Chen, X. Li, and B. Cheng. Degenerated primer design to amplify the heavy chain variable region from immunoglobulin cDNA. *BMC Bioinformatics*, 7(Suppl 4):S9, 2006.