Fast Exact Algorithms for the Closest String and Substring Problems with Application to the Planted (L, d)-Motif Model

Zhi-Zhong Chen and Lusheng Wang

Abstract—We present two parameterized algorithms for the closest string problem. The first runs in $O(nL + nd \cdot 17.97^d)$ time for DNA strings and in $O(nL + nd \cdot 61.86^d)$ time for protein strings, where n is the number of input strings, L is the length of each input string, and d is the given upper bound on the number of mismatches between the center string and each input string. The second runs in $O(nL + nd \cdot 13.92^d)$ time for DNA strings and in $O(nL + nd \cdot 47.21^d)$ time for protein strings. We then extend the first algorithm to a new parameterized algorithm for the closest substring problem that runs in $O((n-1)m^2(L + d \cdot 17.97^d \cdot m^{\lfloor \log_2(d+1) \rfloor}))$ time for DNA strings and in $O((n-1)m^2(L + d \cdot 61.86^d \cdot m^{\lfloor \log_2(d+1) \rfloor}))$ time for protein strings, where n is the number of input strings, L is the length of the center substring, $L - 1 + m$ is the maximum length of a single input string, and d is the given upper bound on the number of mismatches between the center substring and at least one substring of each input string. All the algorithms significantly improve the previous bests. To verify experimentally the theoretical improvements in the time complexity, we implement our algorithm in C and apply the resulting program to the planted (L, d)-motif problem proposed by Pevzner and Sze in 2000. We compare our program with the previously best exact program for the problem, namely PMSPrune (designed by Davila et al. in 2007). Our experimental data show that our program runs faster for practical cases and also for several challenging cases. Our algorithm uses less memory too.

Index Terms—Parameterized algorithm, closest string, closest substring, DNA motif discovery.

1 Introduction

We consider the closest string and the closest substring problems. In the *closest string problem*, we are given a set $\mathcal{S}$ of n strings of equal length L and an integer d (called *radius*). The objective is to compute a string s' of length L such that the Hamming distance $d(s', s)$ between s' and each string $s \in \mathcal{S}$ is at most d. Such a string s' is called a *center string* of the given strings. Of course, center strings may not exist. In that case, an algorithm for the problem should output a special symbol (say, Φ) to indicate this fact.

The *closest substring problem* is a more general problem. In this problem, we are given a set $\mathcal{S}$ of n strings of equal length K and two integers d and L. The objective is to compute a string s' of length L such that each string $s \in \mathcal{S}$ has a substring t of length L with $d(s', t) \leq d$. Note that the letters of substring t appear in string s consecutively. Such a string s' is called a *center substring* of the given strings. Of course, center substrings may not exist. In that case, an algorithm for the problem should output a special symbol (say, Φ) to indicate this fact.

The two problems have been formulated and studied in a variety of applications in bioinformatics and computational biology, such as PCR primer design [6], [10], [12], [16], [22], [24], genetic probe design [12], antisense drug design [5], [12], finding unbiased consensus of a protein family [2], and motif finding [4], [8], [10], [12], [13], [25]. All these applications share a task that requires the design of a new DNA or protein string that is very similar to (a substring of) each of the given strings.

The closest string and substring problems have been proved to be NP-hard [9], [12]. This has motivated researchers to come up with heuristics without performance guarantee [15], [19], [20], parameterized algorithms [7], [11], [17], [23], [26], and polynomial-time approximation algorithms [1], [2], [5], [12], [14] for these problems. We here design parameterized algorithms for them.

Almost all known parameterized algorithms for the closest string problem take d as the parameter. Stojanovic et al. [23] presented a linear-time algorithm for the problem for $d = 1$ only. Gramm et al. [11] designed the first parameterized algorithm for the problem. Their algorithm runs in $O(nL + nd \cdot (d+1)^d)$ time. Ma and Sun [17] improved Gramm et al.'s algorithm to obtain a new parameterized algorithm for the problem that runs in $O(nL + nd \cdot (16(|\Sigma| - 1))^d)$ time, where Σ is the alphabet of the input strings in $\mathcal{S}$. Ma and Sun's improvement is significant because their new algorithm is the first for the problem that can run in polynomial time when the parameter d is logarithmic in the input size. However, the base $16(|\Sigma| - 1)$ of the power $(16(|\Sigma| - 1))^d$ in the time bound of their algorithm is very large. An improved algorithm was given in [26], which runs in $O(nL + nd \cdot (2^{3.25}(|\Sigma| - 1))^d)$ time.

In this paper, we propose two new algorithms for the closest string problem. The first runs in $O(nL + nd \cdot 61.86^d)$ time for protein strings and in $O(nL + nd \cdot 17.97^d)$ time for DNA strings, while the second runs in $O(nL + nd \cdot 47.21^d)$ time for protein strings and in $O(nL + nd \cdot 13.92^d)$ time for DNA strings. In comparison, the previously best algorithm in [26] runs in $O(nL + nd \cdot 180.75^d)$ time for protein strings and in $O(nL + nd \cdot 28.54^d)$ time for DNA strings. We achieve the improvements by a new idea that can be sketched as follows: Like previous algorithms, our algorithm finds a required center string by selecting an arbitrary input string (i.e., a string in $\mathcal{S}$) as the initial candidate center string and gradually modifying at most d letters of the candidate center string so that it becomes a required center string. The modification goes round by round. In each round, to modify the current candidate center string t, the idea is to first find another string $s \in \mathcal{S}$ with $d(t, s) > d$ and then guess at most d positions among the (at most $2d$) positions where t and s have different letters. When using s to modify t, the algorithms in [17] and [26] modify *one or more* letters in t *without looking* at the letters in s, while the algorithm in [11] modifies *only one* letter *by looking* at the letters in s. In contrast, our algorithm modifies *one or more* letters in t *by looking* at the letters in s. In more details, our algorithm guesses the positions i of t where the letter of t at position i should be modified to a letter that differs from both the current letter of t at position i and the letter of s at position i. We then prove a crucial lemma (Lemma 3.1) which says that *not only* d is halved in each round *but also* the more such positions i exist, the faster our algorithm is. We acknowledge that the idea of halving d in each round goes back to Marx [18] and has also been used in [17] and [26]. With the crucial lemma, we then prove a main lemma (Lemma 3.2) which roughly says that the total number of guesses made by our algorithm is at most $\binom{2d}{d}(|\Sigma| + 2\sqrt{|\Sigma| - 1})^d$. This

- Z.-Z. Chen is with the Department of Mathematical Sciences, Tokyo Denki University, Hatomaya, Saitama 350-0394, Japan. E-mail: zzchen@mail.dendai.ac.jp.
- L. Wang is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. Email: cswangl@cityu.edu.hk.

Manuscript received 10 Dec. 2009; revised 7 May 2010; accepted 7 Sept. 2010; published online 27 Jan. 2011.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2009-12-0225. Digital Object Identifier no. 10.1109/TCBB.2011.21.

lemma is a significant improvement over the main theorem (Theorem 1) in [17], which roughly says that the total number of guesses made by the algorithm in [17] is at most $\binom{2d}{d}(4|\Sigma|-4)^d$. Indeed, the proof of our main lemma looks simpler. The bulk of our paper is to show how to obtain an even better bound on the total number of guesses made by our algorithm.

For the closest substring problem, Marx [18] designed a parameterized algorithm whose time complexity is $O(|\Sigma|^{d(\log_2 d+2)} \cdot N^{\log_2 d+O(1)})$, where N is the total length of input strings. Ma and Sun [17] obtained a faster algorithm running in $O(nm^2L + nd \cdot (16(|\Sigma|-1))^d \cdot m^{\lfloor \log_2(d+1)\rfloor+2})$ time with $m = K - L + 1$, by extending their parameterized algorithm for the closest string problem. We can also extend our first algorithm for the closest string problem to an algorithm for the closest substring problem that runs in $O((n-1)m^2(L + d \cdot 17.97^d \cdot m^{\lfloor \log_2(d+1)\rfloor}))$ time for DNA strings and in $O((n-1)m^2(L + d \cdot 61.86^d \cdot m^{\lfloor \log_2(d+1)\rfloor}))$ time for protein strings.

To verify experimentally the theoretical improvements in the time complexity, we implement our algorithm for the closest substring problem in C and apply the resulting program to the following problem proposed in [21]:

Planted (L,d)-Motif Problem: Let M be a fixed but unknown nucleotide sequence (the motif consensus) of length L. Suppose that M occurs once in each of n background sequences of common length K, but that each occurrence of M is corrupted by exactly d point substitutions in positions chosen independently at random. Given the n sequences, recover the motif occurrences and the consensus M.

As in the previous studies [21], [3], we fix $n = 20$ and $K = 600$. We compare our exact program with the previously best exact program called PMSPrune (due to Davila et al. [4]) on randomly generated problem instances. We note in passing that Davila et al. already did experiments to show that PMSPrune is much faster than all the other previously known exact programs for the problem. Our experimental data show that our algorithm runs faster than PMSPrune for practical cases where $(L,d) = (12,3)$, $(13,3)$, $(14,4)$, $(15,4)$, or $(17,5)$, and also runs faster for challenging cases where $(L,d) = (9,2)$ or $(11,3)$. Our algorithm runs slower than PMSPrune for some hard cases where $(L,d) = (17,6)$, $(18,6)$, or $(19,7)$. It should be pointed out that we have found a bug in PMSPrune: For certain problem instances, PMSPrune does not necessarily output *all* solutions. We found the bug by accident: We ran our program and PMSPrune on the same instances, compared their outputs, and happened to find out some solutions that can be output by our program but cannot output by PMSPrune. We also mention in passing that our program uses only $O(nm)$ space while PMSPrune needs $O(nm^2)$ space.

The remainder of this paper is organized as follows: Section 2 contains basic definitions and notations that will be used throughout this paper. Section 3 presents an algorithm for the closest string problem and Section 4 presents another. The former algorithm is easier to understand and can be extended to an algorithm for the closest substring problem. The latter algorithm runs faster but it does not seem to have a natural extension to the closest substring problem. Section 5 extends the former algorithm to the closest substring problem. Section 6 discusses the application to the planted (L,d)-motif problem.

2 Basic Definitions and Notations

Throughout this paper, Σ denotes a fixed alphabet and a string always means one over Σ. For a finite set S (such as Σ), $|S|$ denotes the number of elements in S. Similarly, for a string s, $|s|$ denotes the length of s. A string s has $|s|$ positions, namely, $1, 2, \ldots, |s|$. For convenience, we use $[1..k]$ to denote the set $\{1, 2, \ldots, k\}$. The letter of s at position $i \in [1..|s|]$ is denoted by $s[i]$. Two strings s and t of the same length L *agree* (respectively, *differ*) *at a position* $i \in [1..L]$ if $s[i] = t[i]$ (respectively, $s[i] \neq t[i]$). The *position set where s and t agree* (respectively, *differ*) is the set of all positions $i \in [1..L]$ where s and t agree (respectively, differ).

The following special notations will be very useful. For two or more strings $s_1, \ldots, s_h$ of the same length, $\{s_1 \equiv s_2 \equiv \cdots \equiv s_h\}$ denotes the position set where s_i and s_j agree for all pairs (i,j) with $1 \leq i < j \leq h$, while $\{s_1 \not\equiv s_2 \not\equiv \cdots \not\equiv s_h\}$ denotes the position set where s_i and s_j differ for *all* pairs (i,j) with $1 \leq i < j \leq h$. Moreover, for a sequence $s_1, \ldots, s_h$, $t_1, \ldots, t_k$, $u_1, \ldots, u_\ell$ of strings of the same length with $h \geq 1$, $k \geq 1$, and $\ell \geq 0$, $\{s_1 \equiv s_2 \equiv \cdots \equiv s_h \not\equiv t_1 \not\equiv t_2 \not\equiv \cdots \not\equiv t_k \equiv u_1 \equiv u_2 \equiv \cdots \equiv u_\ell\}$ denotes $\{s_1 \equiv s_2 \equiv \cdots \equiv s_h\} \cap \{s_h \not\equiv t_1 \not\equiv t_2 \not\equiv \cdots \not\equiv t_k\} \cap \{t_k \equiv u_1 \equiv u_2 \equiv \cdots \equiv u_\ell\}$, where $\{s_1\}$ denotes $[1..|s_1|]$ if $h = 1$, while $\{t_k\}$ denotes $[1..|t_k|]$ if $\ell = 0$.

The *hamming distance* between two strings s and t of the same length is $|\{s \not\equiv t\}|$ and is denoted by $d(s,t)$.

3 The First Algorithm

Instead of solving the closest string problem directly, we solve a more general problem called the *extended closest string (ECS) problem*. An instance of the ECS problem is a quintuple $(\mathcal{S}, d, t, P, b)$, where $\mathcal{S}$ is a set of strings of equal length (say, L), t is a string of the same length L, d is a positive integer less than or equal to L, P is a subset of $[1..L]$, and b is a nonnegative integer less than or equal to d. A *solution* to $(\mathcal{S}, d, t, P, b)$ is a string s' of length L satisfying the following conditions:

1. For every position $i \in P$, $s'[i] = t[i]$.
2. The number of positions $i \in [1..L]$ with $s'[i] \neq t[i]$ is at most b.
3. For every string $s \in \mathcal{S}$, $d(s', s) \leq d$.

Intuitively speaking, the first two conditions require that the center string s' be obtained from the candidate string t by modifying at most b letters whose positions in t are outside P. Given an instance $(\mathcal{S}, d, t, P, b)$, the ECS problem asks to output a solution if one exists. The output has to be a special symbol (say, Φ) if no solution exists.

Obviously, to solve the closest string problem for a given instance $(\mathcal{S}, d)$, it suffices to solve the ECS problem for the instance $(\mathcal{S}, d, s, \emptyset, d)$, where s is an arbitrary string in $\mathcal{S}$ and $\emptyset$ is the empty set. That is, we can solve the closest string problem by calling any algorithm for the ECS problem once. So, we hereafter focus on the ECS problem instead of the closest string problem.

Intuitively speaking, given an instance $(\mathcal{S}, d, t, P, b)$, the ECS problem asks us to modify the letters of at most b positions (outside P) of t so that t becomes a string s' with $d(s', s) \leq d$ for every string $s \in \mathcal{S}$. A naive way is to first guess at most b positions among the $L - |P|$ positions (outside P) of t and then modify the letters at the guessed positions. A better idea has been used in the algorithms in [17] and [26]: First, try to find a string $s \in \mathcal{S}$ with $d(t,s) > d$ and then use s to help guess the (at most b) positions of t where the

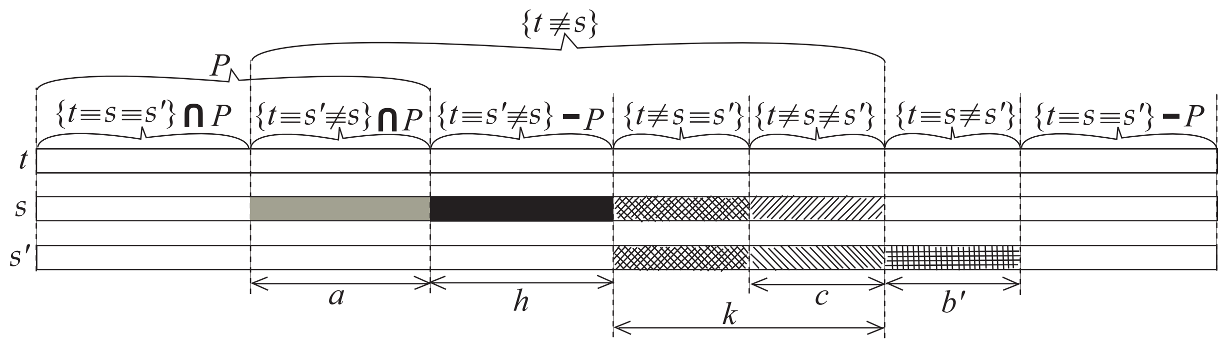
$\{t \not\equiv s\}$
P
$\{t \equiv s \equiv s'\} \cap P$
$\{t \equiv s' \not\equiv s\} \cap P$
$\{t \equiv s' \not\equiv s\} - P$
$\{t \not\equiv s \equiv s'\}$
$\{t \not\equiv s \not\equiv s'\}$
$\{t \equiv s \not\equiv s'\}$
$\{t \equiv s \equiv s'\} - P$
t
s
s'
a
h
k
c
b'

Fig. 1. Strings t, s, and s' in Lemma 3.1, where for each position $i \in [1..L]$, two of the strings have the same letter at position i if and only if they are illustrated in the same color or pattern at position i.

letters of t should be modified. For each guessed position i, the algorithms in [17] and [26] try all possible ways to modify $t[i]$. Note that there are $|\Sigma| - 1$ possible ways to modify $t[i]$. So, there can be $(|\Sigma| - 1)^b$ possible ways to modify t.

Our new observation is that there may be some guessed positions i such that $t[i]$ may be changed to $s[i]$. In other words, for such a position i, we do not have to guess a new letter for $t[i]$. This can save us a lot of time, as justified by the following lemma:

Lemma 3.1. *Let $(\mathcal{S}, d, t, P, b)$ be an instance of the ECS problem. Assume that s' is a solution to $(\mathcal{S}, d, t, P, b)$. Let s be a string in $\mathcal{S}$, let $\ell = d(t, s) - d$, let k be the number of positions $i \in \{t \not\equiv s\} - P$ with $s'[i] \neq t[i]$, let c be the number of positions $i \in \{t \not\equiv s\} - P$ with $s'[i] \neq t[i]$ and $s'[i] \neq s[i]$, and let b' be the number of positions $i \in [1..L] - (P \cup \{t \not\equiv s\})$ with $s'[i] \neq t[i]$. Then, $b' \leq \min\{b - k, k - \ell - c\}$. Consequently, $b' \leq \frac{b - \ell - c}{2}$.*

Proof. Obviously, $d(t, s') = k + b'$. Since s' is a solution to instance $(\mathcal{S}, d, t, P, b)$, $d(t, s') \leq b$. Thus, $b' \leq b - k$.

Let $a = |P \cap \{t \not\equiv s\}|$, and let h be the number of positions $i \in \{t \not\equiv s\} - P$ with $t[i] = s'[i]$ (see Fig. 1). Then, $|\{t \not\equiv s\}| = a + h + k$ and $d(s', s) = a + h + c + b'$. So, by assumption, $a + h + k = d + \ell$ and $a + h + c + b' \leq d$. Thus, $b' \leq k - \ell - c$. □

We note that Lemma 3.1 is stronger than Lemma 1 in [17] and Lemma 2 in [26].

Based on Lemma 3.1, we now design an algorithm called **CloseString** for the ECS problem:

Algorithm 1. CloseString

```
Input: An instance $(\mathcal{S}, d, t, P, b)$ of the ECS problem.
Output: A solution to $(\mathcal{S}, d, t, P, b)$ if one exists, or $\Phi$ otherwise.
1. If there is no $s \in \mathcal{S}$ with $d(t, s) > d$, then output $t$ and halt.
2. Find a string $s \in \mathcal{S}$ with $d(t, s) > d$.
3. Let $\ell = d(t, s) - d$ and $R = \{t \not\equiv s\} - P$.
4. If $\ell > \min\{b, |R|\}$, then return $\Phi$.
5. Make a copy $t'$ of $t$.
6. For each subset $X$ of $R$ with $\ell \leq |X| \leq b$ and for each subset $Y$
   of $X$ with $|Y| \leq |X| - \ell$, perform the following steps:
      6.1. For each position $i \in X - Y$, change $t[i]$ to $s[i]$.
      6.2. For all $(|\Sigma| - 2)^{|Y|}$ possible ways to change the letters
           of $t$ at the positions in $Y$ (so that the letter of $t$ at each
           position $i \in Y$ is changed to a letter other than $s[i]$
           and $t[i]$), change the letters of $t$ at the $|Y|$ positions
           and then call CloseString($\mathcal{S} - \{s\}, d, t, P \cup R,$
           $\min\{b - |X|, |X| - \ell - |Y|\}$) recursively.
      6.3. Restore $t$ back to $t'$.
7. Return $\Phi$.
```

To see the correctness of our algorithm, first observe that Step 1 is clearly correct. To see that Step 4 is also correct, first note that $d(t, s) = |\{t \not\equiv s\}| = d + \ell$. So, in order to satisfy $d(t, s) \leq d$, we need to first select at least ℓ positions among the positions in $\{t \not\equiv s\}$ and then modify the letters at the selected positions. By definition, we are allowed to select at most b positions and the selected positions have to be in $R = \{t \not\equiv s\} - P$; so no solution exists if $\ell > \min\{b, |R|\}$. The correctness of Step 6.2 is guaranteed by Lemma 3.1. This can be seen by viewing $|X|$ in the algorithm as k in Lemma 3.1, viewing $|Y|$ in the algorithm as c in Lemma 3.1, and viewing b' in Lemma 3.1 as the number of positions (outside $P \cup \{t \not\equiv s\} = P \cup R$) of t where the letters have to be modified in order to transform t into a solution. That is, $\min\{b - |X|, |X| - \ell - |Y|\}$ in Step 6.2 corresponds exactly to $\min\{b - k, k - \ell - c\}$ in Lemma 3.1.

The remainder of this section is devoted to estimating the running time of the algorithm. The execution of algorithm **CloseString** on input $(\mathcal{S}, d, t, P, b)$ can be modeled by a tree $\mathcal{T}$ in which the root corresponds to $(\mathcal{S}, d, t, P, b)$, each other node corresponds to a recursive call, and a recursive call A is a child of another call B if and only if B calls A directly. We call $\mathcal{T}$ the *search tree* on input $(\mathcal{S}, d, t, P, b)$. By the construction of algorithm **CloseString**, each nonleaf node in $\mathcal{T}$ has at least two children. Thus, the number of nodes in $\mathcal{T}$ is at most twice the number of leaves in $\mathcal{T}$. Consequently, we can focus on how to bound the number of leaves in $\mathcal{T}$. For convenience, we define the *size* of $\mathcal{T}$ to be the number of its leaves. The following lemma shows an upper bound on the size of $\mathcal{T}$:

Lemma 3.2. *Let $T(d, b)$ be the size of the search tree on input $(\mathcal{S}, d, t, P, b)$. Then,*

$$T(d, b) \leq \binom{d + b}{b} \left(|\Sigma| + 2\sqrt{|\Sigma| - 1}\right)^b.$$

Proof. By induction on b. In case $b = 0$, the algorithm will output either t or Φ without making a recursive call; so, $T(d, 0) = 1$ and the lemma holds. Similarly, in case b is small enough that the algorithm does not make a recursive call, we have $T(d, b) = 1$ and the lemma holds. So, assume that b is large enough that the algorithm makes at least one recursive call. Then, by the algorithm, we have the following inequality:

$$T(d, b) \leq \sum_{k=\ell}^{b} \binom{d + \ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma| - 2)^c \, T(d, \min\{b - k, k - \ell - c\}). \quad (1)$$

Let $m = \min\{b-k, k-\ell-c\}$. Then, by the induction hypothesis

$$T(d,b) \le \sum_{k=\ell}^{b} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \binom{d+m}{m} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^m. \tag{2}$$

Since $m \le b-k \le d$, $\binom{d+m}{m} \le \binom{d+b-k}{b-k}$. Moreover, since $m \le \frac{b-\ell-c}{2}$ and

$$|\Sigma| + 2\sqrt{|\Sigma|-1} = \left(1+\sqrt{|\Sigma|-1}\right)^2,$$
$$\left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^m \le \left(1+\sqrt{|\Sigma|-1}\right)^{b-\ell-c}.$$

So, by Inequality (2), we have

$$\begin{aligned}
T(d,b) &\le \sum_{k=\ell}^{b} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \binom{d+b-k}{b-k} \left(1+\sqrt{|\Sigma|-1}\right)^{b-\ell-c} \\
&= \left(1+\sqrt{|\Sigma|-1}\right)^{b-\ell} \sum_{k=\ell}^{b} \binom{d+\ell}{k} \binom{d+b-k}{b-k} \sum_{c=0}^{k-\ell} \binom{k}{c} \left(\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^c \\
&\le \left(1+\sqrt{|\Sigma|-1}\right)^{b-\ell} \sum_{k=\ell}^{b} \binom{d+b}{k} \binom{d+b-k}{b-k} \left(1+\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^k \\
&= \left(1+\sqrt{|\Sigma|-1}\right)^{b-\ell} \binom{d+b}{b} \sum_{k=\ell}^{b} \binom{b}{k} \left(1+\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^k \\
&\le \left(1+\sqrt{|\Sigma|-1}\right)^{b} \binom{d+b}{b} \left(2+\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^b \\
&= \binom{d+b}{b} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^b,
\end{aligned} \tag{3}$$

where Inequality (3) follows from the binomial theorem and the fact that $\ell \le b$. □

We note that Lemma 3.2 is much stronger than Theorem 1 in [17], which seems to be the main result in [17]. In particular, when $b = d$, our upper bound $\binom{2d}{d}(|\Sigma| + 2\sqrt{|\Sigma|-1})^d$ is already better than the upper bound $\binom{2d}{d}(|\Sigma|-1)^d\, 2^{1.25d}$ in [26], if $|\Sigma| \ge 5$.

We next show an even better upper bound on $T(d,d)$. The idea is to distinguish the first recursive call of algorithm **CloseString** with the other recursive calls. With this idea, the parameter ℓ comes into play when we analyze $T(d,d)$. First, we need the following technical lemma:

Lemma 3.3. *Suppose that α is a real number with $0 < \alpha \le \frac{1}{2}$ and m is a positive integer with $1 \le \lfloor \alpha m \rfloor \le m-1$. Then, $\binom{m}{\lfloor \alpha m \rfloor} \le \alpha^{-\alpha m}(1-\alpha)^{-(1-\alpha)m}$.*

Proof. By Stirling's formula, for any integer $h \ge 0$, $h! = \sqrt{2\pi h}\left(\frac{h}{e}\right)^h e^{\lambda_h}$ with $\frac{1}{12h+1} < \lambda_h < \frac{1}{12h}$. So,

$$\begin{aligned}
\binom{m}{\lfloor \alpha m \rfloor} &= \frac{m!}{(\lfloor \alpha m \rfloor)!\,(m-\lfloor \alpha m \rfloor)!} \\
&\le \left\{\sqrt{2\pi m}\left(\frac{m}{e}\right)^m e^{\frac{1}{12m}}\right\} \Big/ \left\{\sqrt{2\pi\lfloor \alpha m \rfloor}\left(\frac{\lfloor \alpha m \rfloor}{e}\right)^{\lfloor \alpha m \rfloor} e^{\frac{1}{12\lfloor \alpha m \rfloor+1}} \right. \\
&\quad \left. \sqrt{2\pi(m-\lfloor \alpha m \rfloor)}\left(\frac{m-\lfloor \alpha m \rfloor}{e}\right)^{m-\lfloor \alpha m \rfloor} e^{\frac{1}{12(m-\lfloor \alpha m \rfloor)+1}}\right\} \\
&\le \frac{m^m}{\lfloor \alpha m \rfloor^{\lfloor \alpha m \rfloor}(m-\lfloor \alpha m \rfloor)^{m-\lfloor \alpha m \rfloor}}.
\end{aligned}$$

The last inequality holds because the assumption $1 \le \lfloor \alpha m \rfloor \le m-1$ guarantees that $2\pi\lfloor \alpha m \rfloor(m-\lfloor \alpha m \rfloor) \ge m$ and $12\lfloor \alpha m \rfloor + 1 \le 12m$.

Now, since $\alpha \le \frac{1}{2}$ and the function $f(x) = x^x(m-x)^{m-x}$ is decreasing when $0 \le x \le \frac{m}{2}$, we have

$$\binom{m}{\lfloor \alpha m \rfloor} \le \frac{m^m}{(\alpha m)^{\alpha m}(m-\alpha m)^{m-\alpha m}} = \alpha^{-\alpha m}(1-\alpha)^{-(1-\alpha)m}.$$

□

Now, we are ready to show the main theorem of this section:

Theorem 3.4. $T(d,d) \le (1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2})^d$.

Proof. As mentioned in the proof of Lemma 3.2, if our algorithm makes no recursive calls, then $T(d,d) = 1$ and hence the theorem clearly holds. So, assume that our algorithm makes at least one recursive call. Then, by Inequality (1), we have

$$T(d,d) \le \sum_{k=\ell}^{d} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c\, T(d, \min\{d-k, k-\ell-c\}). \tag{4}$$

Using Inequality (4) and the fact that $T(d,0) \le 1$, one can easily verify that $T(1,1) \le 2$ and $T(2,2) \le 6|\Sigma| - 6$. Since

$$2 \le 1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2} \quad \text{and}$$
$$6|\Sigma| - 6 \le \left(1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2}\right)^2,$$

the theorem holds when $d \le 2$. So, in the sequel, we assume that $d \ge 3$.

For convenience, let $b' = \min\{d-k, k-\ell-c\}$. Then, by Lemma 3.2

$$T(d,d) \le \sum_{k=\ell}^{d} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \binom{d+b'}{b'} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^{b'}. \tag{5}$$

Since $b' \le \frac{d-\ell-c}{2}$, $b' \le \lfloor \frac{d+b'}{3} \rfloor$ and in turn $\binom{d+b'}{b'} \le \binom{d+b'}{\lfloor \frac{d+b'}{3} \rfloor}$. Moreover, since $d \ge 3$, $1 \le \lfloor \frac{d+b'}{3} \rfloor \le d + b' - 1$. So, fixing $\alpha = \frac{1}{3}$ in Lemma 3.3, we have

$$\binom{d+b'}{b'} \le \binom{d+b'}{\lfloor \frac{d+b'}{3} \rfloor} \le \left(\frac{1}{3}\right)^{-\frac{d+b'}{3}} \left(\frac{2}{3}\right)^{-\frac{2(d+b')}{3}} = \left(\frac{3}{\sqrt[3]{4}}\right)^{d+b'}.$$

TABLE 1
Comparison of Bases of the Powers in the Time Bounds

	$\lvert\Sigma\rvert = 4$ (DNA)	$\lvert\Sigma\rvert = 20$ (protein)
$16(\lvert\Sigma\rvert - 1)$ [17]	48	304
$2^{3.25}(\lvert\Sigma\rvert - 1)$ [26]	28.54...	180.75...
$1.5\sqrt[3]{2}\lvert\Sigma\rvert + 3\sqrt{3(\lvert\Sigma\rvert - 1)} + 3\sqrt{3} - 3\sqrt[3]{2}$ [this paper]	17.97...	61.86...

Hence, by Inequality (5), we have

$$
\begin{aligned}
T(d,d) & \\
&\le \sum_{k=\ell}^{d}\binom{d+\ell}{k}\sum_{c=0}^{k-\ell}\binom{k}{c}(|\Sigma|-2)^c\left(\frac{3}{\sqrt[3]{4}}\right)^{d+b'}\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)^{b'}\\
&\le \sum_{k=\ell}^{d}\binom{d+\ell}{k}\sum_{c=0}^{k-\ell}\binom{k}{c}(|\Sigma|-2)^c\left(\frac{3}{\sqrt[3]{4}}\right)^{d+\frac{d-\ell-c}{2}}\\
&\quad\left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell-c} \qquad (6)\\
&= \sum_{k=\ell}^{d}\binom{d+\ell}{k}\sum_{c=0}^{k-\ell}\binom{k}{c}\left(\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^c\left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell-c}\\
&\quad\left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell}\\
&= \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell}\left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell}\sum_{k=\ell}^{d}\binom{d+\ell}{k}\\
&\quad\sum_{c=0}^{k-\ell}\binom{k}{c}\left(\frac{\sqrt[3]{2}(|\Sigma|-2)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}\right)^c\\
&\le \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell}\left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell}\sum_{k=\ell}^{d}\binom{d+\ell}{k}\\
&\quad\left(1+\frac{\sqrt[3]{2}(|\Sigma|-2)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}\right)^k \qquad (7)\\
&\le \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell}\left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell}\left(2+\frac{\sqrt[3]{2}(|\Sigma|-2)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}\right)^{d+\ell} \qquad (8)\\
&= \left(1.5\sqrt[3]{2}|\Sigma|+3\sqrt{3(|\Sigma|-1)}+3\sqrt{3}-3\sqrt[3]{2}\right)^d\\
&\quad\cdot\left(\frac{\sqrt[3]{2}\left(2\sqrt{3}+2\sqrt{3(|\Sigma|-1)}+\sqrt[3]{2}|\Sigma|-2\sqrt[3]{2}\right)}{3\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)}\right)^\ell\\
&\le \left(1.5\sqrt[3]{2}|\Sigma|+3\sqrt{3(|\Sigma|-1)}+3\sqrt{3}-3\sqrt[3]{2}\right)^d, \qquad (9)
\end{aligned}
$$

where Inequality (6) holds for $b' \le \frac{d-\ell-c}{2}$ and $|\Sigma|+2\sqrt{|\Sigma|-1} = (1+\sqrt{|\Sigma|-1})^2$, Inequalities (7) and (8) follow from the binomial theorem, and Inequality (9) follows from the fact that

$$\sqrt[3]{2}\left(2\sqrt{3}+2\sqrt{3(|\Sigma|-1)}+\sqrt[3]{2}|\Sigma|-2\sqrt[3]{2}\right) \le 3\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)$$

for $|\Sigma| \ge 2$. This finishes the proof. □

Corollary 3.5. *Algorithm* ***CloseString*** *solves the ECS problem in time*

$$O\left(nL + nd\cdot\left(1.5\sqrt[3]{2}|\Sigma|+3\sqrt{3(|\Sigma|-1)}+3\sqrt{3}-3\sqrt[3]{2}\right)^d\right).$$

Proof. Obviously, each leaf of the search tree takes $O(nL)$ time. As observed in previous works (e.g., [17]), we can improve this time bound by carefully remembering the previous distances and only updating the $O(d)$ positions changed. The conclusion is this: With an $O(nL)$-time preprocessing, each leaf of the search tree takes $O(nd)$ time. So, by Theorem 3.4, the total time complexity of algorithm **CloseString** is as stated in the corollary. □

The best algorithm in [26] for the closest string problem runs in $O(nL + nd\cdot(2^{3.25}(|\Sigma|-1))^d)$ time, while the algorithm in [17] for the same problem runs in $O(nL + nd\cdot(16(|\Sigma|-1))^d)$ time. For these two previous algorithms as well as our new algorithm **CloseString**, their time bounds contain a power whose exponent is d. Table 1 shows a comparison of the bases of the powers.

4 The Second Algorithm

Motivated by an idea in [26], we obtain the second algorithm by modifying Step 2 of the algorithm in Section 3 as follows:

2. Find a string $s \in \mathcal{S}$ such that $d(t,s)$ is maximized over all strings in $\mathcal{S}$.

We call the modified algorithm **CloseString2**. The intuition behind **CloseString2** is this: By Lemma 3.1, the larger ℓ is, the smaller b' is. Note that b' means the number of letters of t we need to further modify. Thus, by maximizing ℓ, we can make our algorithm run faster.

Throughout the remainder of this section, fix an input $(\mathcal{S},d,t,P,b)$ to algorithm **CloseString2** and consider the search tree $\mathcal{T}$ of **CloseString2** on this input. Let r denote the root of $\mathcal{T}$.

During the execution of **CloseString2** on input $(\mathcal{S},d,t,P,b)$, d does not change but the other parameters may change. That is, each node of $\mathcal{T}$ corresponds to a recursive call whose input is of the form $(\mathcal{S}',d,t',P',b')$, where $\mathcal{S}'$ is a subset of $\mathcal{S}$, t' is a modification of t, P' is a superset of P, and b' is an integer smaller than b. So, for each node u of $\mathcal{T}$, we use $\mathcal{S}_u$, t_u, P_u, and b_u to denote the first, the third, the fourth, and the fifth parameter, respectively, in the input given to the recursive call corresponding to u. For example, $\mathcal{S}_r = \mathcal{S}$, $t_r = t$, $P_r = P$, and $b_r = b$. Moreover, for each node u of $\mathcal{T}$, we use s_u and ℓ_u to denote the string s and the integer ℓ computed in Steps 2 and 3 of the recursive call corresponding to u, respectively.

Obviously, if the set R computed in Step 3 of the algorithm is small, then the number of subsets X tried in Step 6 should be small. Intuitively speaking, the next lemma shows that $|R|$ cannot be so large.

Lemma 4.1. *Suppose that u is a nonleaf descendant of r in $\mathcal{T}$. Then, $|P_u \cap \{t_u \not\equiv s_u\}| \ge \frac{d(t_u,t_r)-\ell_r+\ell_u}{2}$.*

Proof. For ease of explanation, we define the following notations (cf. Fig. 2):

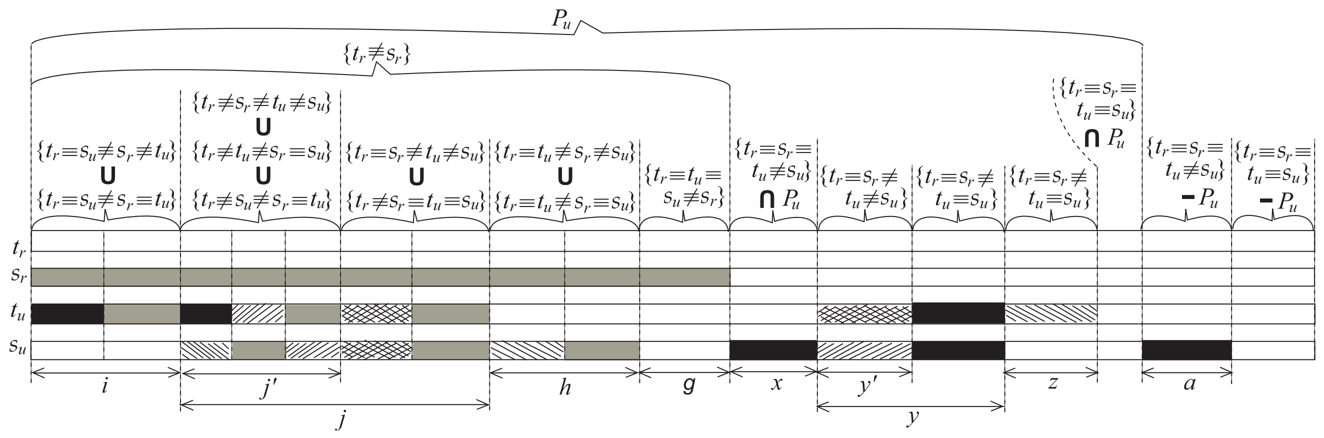
P_u
$\{t_r \not\equiv s_r\}$
$\{t_r \equiv s_u \not\equiv s_r \not\equiv t_u\} \cup \{t_r \equiv s_u \not\equiv s_r \equiv t_u\}$
$\{t_r \not\equiv s_r \not\equiv t_u \not\equiv s_u\} \cup \{t_r \not\equiv t_u \not\equiv s_r \equiv s_u\} \cup \{t_r \not\equiv s_u \not\equiv s_r \equiv t_u\}$
$\{t_r \equiv s_r \not\equiv t_u \not\equiv s_u\} \cup \{t_r \not\equiv s_r \equiv t_u \equiv s_u\}$
$\{t_r \equiv t_u \not\equiv s_r \not\equiv s_u\} \cup \{t_r \equiv t_u \not\equiv s_r \equiv s_u\}$
$\{t_r \equiv t_u \equiv s_u \not\equiv s_r\}$
$\{t_r \equiv s_r \equiv t_u \not\equiv s_u\} \cap P_u$
$\{t_r \equiv s_r \not\equiv t_u \not\equiv s_u\}$
$\{t_r \equiv s_r \not\equiv t_u \equiv s_u\}$
$\{t_r \equiv s_r \not\equiv t_u \equiv s_u\}$
$\{t_r \equiv s_r \equiv t_u \equiv s_u\} \cap P_u$
$\{t_r \equiv s_r \equiv t_u \not\equiv s_u\} - P_u$
$\{t_r \equiv s_r \equiv t_u \equiv s_u\} - P_u$
t_r
s_r
t_u
s_u
i
j'
h
g
x
y'
z
a
j
y

Fig. 2. Strings t_r, s_r, t_u, and s_u in Lemma 4.1, where for each position $i \in [1..L]$, two of the strings have the same letter at position i if and only if they are illustrated in the same color or pattern at position i.

- $i = |\{t_r \equiv s_u \not\equiv s_r \not\equiv t_u\}| + |\{t_r \equiv s_u \not\equiv s_r \equiv t_u\}|$.
- $j' = |\{t_r \not\equiv s_r \not\equiv t_u \not\equiv s_u\}| + |\{t_r \not\equiv t_u \not\equiv s_r \equiv s_u\}| + |\{t_r \not\equiv s_u \not\equiv s_r \equiv t_u\}|$.
- $j = j' + |\{t_r \equiv s_r \not\equiv t_u \not\equiv s_u\}| + |\{t_r \not\equiv s_r \equiv t_u \equiv s_u\}|$.
- $h = |\{t_r \equiv t_u \not\equiv s_r \not\equiv s_u\}| + |\{t_r \equiv t_u \not\equiv s_r \equiv s_u\}|$.
- $g = |\{t_r \equiv t_u \equiv s_u \not\equiv s_r\}|$.
- $x = |\{t_r \equiv s_r \equiv t_u \not\equiv s_u\} \cap P_u|$.
- $y' = |\{t_r \equiv s_r \not\equiv t_u \not\equiv s_u\}|$.
- $y = y' + |\{t_r \equiv s_r \not\equiv t_u \equiv s_u\}|$.
- $z = |\{t_r \equiv s_r \not\equiv t_u \equiv s_u\}|$.
- $a = |\{t_r \equiv s_r \equiv t_u \not\equiv s_u\} - P_u|$.

By simply counting the number of positions where t_r and s_r differ, we have the following equation immediately:

$$d(t_r, s_r) = i + j + h + g = d + \ell_r. \tag{10}$$

Similarly, we have the following four equations:

$$d(t_r, s_u) = j + h + x + y + a \tag{11}$$
$$d(t_u, s_u) = i + j' + h + x + y' + z + a = d + \ell_u \tag{12}$$
$$d(t_r, t_u) = i + j + y + z \tag{13}$$
$$|P_u \cap \{t_u \not\equiv s_u\}| = i + j' + h + x + y' + z. \tag{14}$$

By Step 2 of algorithm **CloseString2**, $d(t_r, s_u) \le d(t_r, s_r)$. So, (10) and (11) imply the following inequality:

$$x + y + a \le i + g. \tag{15}$$

By (13) and (14), in order to finish the proof, we need only to prove the following inequality:

$$i + j' + h + x + y' + z \ge \frac{i + j - \ell_r + y + z + \ell_u}{2}. \tag{16}$$

By (10), Inequality (16) is equivalent to the following inequality:

$$2i + 2j' + 3h + 2x + 2y' + z + g \ge d + \ell_u + y. \tag{17}$$

By (12), Inequality (17) is equivalent to the following inequality:

$$i + j' + 2h + x + y' + g \ge a + y. \tag{18}$$

By Inequality (15), Inequality (18) holds. This finishes the proof. □

We note in passing that there is a lemma in [26] similar to but much weaker than Lemma 4.1.

The following lemma is similar to Lemma 3.2:

Lemma 4.2. *Suppose that r has at least one child in $\mathcal{T}$. For each descendant u of r in $\mathcal{T}$, let $F(d, b_u)$ denote the number of nodes in the subtree of $\mathcal{T}$ rooted at u. Then, for each descendant u of r in $\mathcal{T}$,*

$$F(d, b_u) \le \binom{\lfloor \frac{2d - d(t_u, t_r) + \ell_r + b_u}{2} \rfloor}{b_u} \left(|\Sigma| + 2\sqrt{|\Sigma| - 1}\right)^{b_u}.$$

Proof. The proof is similar to that of Lemma 3.2 and is by induction on b_u. In case b_u is small enough that algorithm **CloseString2** on input $(\mathcal{S}_u, d, t_u, P_u, b_u)$ does not make a recursive call, we have $F(d, b_u) = 1$ and the lemma holds. So, assume that b_u is large enough that algorithm **CloseString2** on input $(\mathcal{S}_u, d, t_u, P_u, b_u)$ makes at least one recursive call. Then, by the algorithm and Lemma 4.1

$$\begin{aligned} F(d, b_u) &\le \sum_{k=\ell_u}^{b_u} \binom{d + \ell_u - \lceil \frac{d(t_u, t_r) - \ell_r + \ell_u}{2} \rceil}{k} \\ &\quad \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma| - 2)^c F(d, \min\{b_u - k, k - \ell_u - c\}) \\ &= \sum_{k=\ell_u}^{b_u} \binom{\lfloor \frac{2d - d(t_u, t_r) + \ell_r + \ell_u}{2} \rfloor}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} \\ &\quad (|\Sigma| - 2)^c F(d, \min\{b_u - k, k - \ell_u - c\}). \end{aligned} \tag{19}$$

Let $h = \lfloor \frac{2d - d(t_u, t_r) + \ell_r + b_u}{2} \rfloor$ and $m = \min\{b_u - k, k - \ell_u - c\}$. Note that $\ell_u \le b_u$ and k means the number of positions $i \in [1..L]$ such that $t_u[i]$ is changed by algorithm **CloseString2** on input $(\mathcal{S}_u, d, t_u, P_u, b_u)$. Now, by Inequality (19) and the induction hypothesis,

$$\begin{aligned} F(d, b_u) &\le \sum_{k=\ell_u}^{b_u} \binom{h}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma| - 2)^c \binom{\lfloor \frac{2d - d(t_u, t_r) - k + \ell_r + m}{2} \rfloor}{m} \\ &\quad \left(|\Sigma| + 2\sqrt{|\Sigma| - 1}\right)^m. \end{aligned} \tag{20}$$

By Lemma 3.1, $b_u \le \frac{b_r}{2}$. So, $b_u \le \frac{d}{2}$ for $b_r \le d$. We also have $d(t_u, t_r) + b_u \le b_r$. Using these facts, one can verify that $\frac{2d - d(t_u, t_r) - 2k + \ell_r + b_u}{2} \ge 2(b_u - k)$. Thus, by the fact that $m \le b_u - k$, we have

$$\begin{aligned}F(d,b_u) &\le \sum_{k=\ell_u}^{b_u}\binom{h}{k}\sum_{c=0}^{k-\ell_u}\binom{k}{c}(|\Sigma|-2)^c\binom{\left\lfloor\frac{2d-d(t_u,t_r)-2k+\ell_r+b_u}{2}\right\rfloor}{b_u-k}\\ &\quad\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)^m\\ &=\sum_{k=\ell_u}^{b_u}\binom{h}{k}\sum_{c=0}^{k-\ell_u}\binom{k}{c}(|\Sigma|-2)^c\binom{h-k}{b_u-k}\\ &\quad\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)^m.\end{aligned}\tag{21}$$

Since $m \le \frac{b_u-\ell_u-c}{2}$ and $|\Sigma|+2\sqrt{|\Sigma|-1}=(1+\sqrt{|\Sigma|-1})^2$, Inequality (21) implies the following:

$$\begin{aligned}F(d,b_u) &\le \sum_{k=\ell_u}^{b_u}\binom{h}{k}\sum_{c=0}^{k-\ell_u}\binom{k}{c}(|\Sigma|-2)^c\binom{h-k}{b_u-k}\\ &\quad\left(1+\sqrt{|\Sigma|-1}\right)^{b_u-\ell_u-c}\\ &=\left(1+\sqrt{|\Sigma|-1}\right)^{b_u-\ell_u}\sum_{k=\ell_u}^{b_u}\binom{h}{k}\binom{h-k}{b_u-k}\\ &\quad\sum_{c=0}^{k-\ell_u}\binom{k}{c}\left(\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^c\\ &\le\left(1+\sqrt{|\Sigma|-1}\right)^{b_u-\ell_u}\sum_{k=\ell_u}^{b_u}\binom{h}{k}\binom{h-k}{b_u-k}\\ &\quad\left(1+\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^k \qquad (22)\\ &=\left(1+\sqrt{|\Sigma|-1}\right)^{b_u-\ell_u}\binom{h}{b_u}\sum_{k=\ell_u}^{b_u}\binom{b_u}{k}\left(1+\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^k\\ &\le\left(1+\sqrt{|\Sigma|-1}\right)^{b_u}\binom{h}{b_u}\left(2+\frac{|\Sigma|-2}{1+\sqrt{|\Sigma|-1}}\right)^{b_u} \qquad (23)\\ &=\binom{h}{b_u}\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)^{b_u},\end{aligned}$$

where Inequalities (22) and (23) follow from the binomial theorem. This completes the proof. □

Now, we are ready to prove the main theorem in this section:

Theorem 4.3. *If $|\Sigma|=2$, then the size $T(d,d)$ of tree $\mathcal{T}$ is at most 8^d. Otherwise,*

$$T(d,d)\le\left(\sqrt{2}|\Sigma|+\sqrt[4]{8}\left(\sqrt{2}+1\right)\left(1+\sqrt{|\Sigma|-1}\right)-2\sqrt{2}\right)^d.$$

Proof. The proof is similar to that of Theorem 3.4. If our algorithm makes no recursive calls, then $T(d,d)=1$ and hence the theorem clearly holds. So, assume that our algorithm makes at least one recursive call. Then, by the algorithm, we have

$$\begin{aligned}T(d,d)&\le\sum_{k=\ell_r}^{d}\binom{d+\ell_r}{k}\sum_{c=0}^{k-\ell_r}\binom{k}{c}(|\Sigma|-2)^c\\ &\quad F(d,\min\{d-k,k-\ell_r-c\}).\end{aligned}\tag{24}$$

For convenience, let $m=\min\{d-k,k-\ell_r-c\}$. Then, by Lemma 4.2,

$$\begin{aligned}T(d,d)&\le\sum_{k=\ell_r}^{d}\binom{d+\ell_r}{k}\sum_{c=0}^{k-\ell_r}\binom{k}{c}(|\Sigma|-2)^c\binom{\left\lfloor\frac{2d-k+\ell_r+m}{2}\right\rfloor}{m}\\ &\quad\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)^m\\ &\le\sum_{k=\ell_r}^{d}\binom{d+\ell_r}{k}\sum_{c=0}^{k-\ell_r}\binom{k}{c}(|\Sigma|-2)^c\,2^{\frac{2d-k+\ell_r+m}{2}}\\ &\quad\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)^m\\ &=\sum_{k=\ell_r}^{d}2^{\frac{2d-k+\ell_r}{2}}\binom{d+\ell_r}{k}\sum_{c=0}^{k-\ell_r}\binom{k}{c}(|\Sigma|-2)^c\\ &\quad\left(\sqrt{2}\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)\right)^m.\end{aligned}\tag{25}$$

Note that $m\le\frac{d-\ell_r-c}{2}$ and $|\Sigma|+2\sqrt{|\Sigma|-1}=(1+\sqrt{|\Sigma|-1})^2$. So, by Inequality (25), we have

$$\begin{aligned}T(d,d)&\le\sum_{k=\ell_r}^{d}2^{\frac{2d-k+\ell_r}{2}}\binom{d+\ell_r}{k}\sum_{c=0}^{k-\ell_r}\binom{k}{c}(|\Sigma|-2)^c\\ &\quad\left(\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)\right)^{d-\ell_r-c}\\ &=\sum_{k=\ell_r}^{d}2^{\frac{2d-k+\ell_r}{2}}\binom{d+\ell_r}{k}\left(\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)\right)^{d-\ell_r}\sum_{c=0}^{k-\ell_r}\binom{k}{c}\\ &\quad\left(\frac{|\Sigma|-2}{\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)}\right)^c\\ &\le\sum_{k=\ell_r}^{d}2^{\frac{2d-k+\ell_r}{2}}\binom{d+\ell_r}{k}\left(\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)\right)^{d-\ell_r}\\ &\quad\left(1+\frac{|\Sigma|-2}{\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)}\right)^k \qquad (26)\\ &=2^{d+\frac{\ell_r}{2}}\left(\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)\right)^{d-\ell_r}\sum_{k=\ell_r}^{d}\binom{d+\ell_r}{k}\\ &\quad\left(\frac{1}{\sqrt{2}}+\frac{|\Sigma|-2}{\sqrt[4]{8}\left(1+\sqrt{|\Sigma|-1}\right)}\right)^k\\ &\le 2^{d+\frac{\ell_r}{2}}\left(\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)\right)^{d-\ell_r}\\ &\quad\left(1+\frac{1}{\sqrt{2}}+\frac{|\Sigma|-2}{\sqrt[4]{8}\left(1+\sqrt{|\Sigma|-1}\right)}\right)^{d+\ell_r} \qquad (27)\\ &=\left(\sqrt{2}|\Sigma|+\sqrt[4]{8}\left(\sqrt{2}+1\right)\left(1+\sqrt{|\Sigma|-1}\right)-2\sqrt{2}\right)^d\\ &\quad\left(\frac{\left(\sqrt[4]{8}+\sqrt[4]{2}\right)\left(1+\sqrt{|\Sigma|-1}\right)+|\Sigma|-2}{\sqrt{2}\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right)}\right)^{\ell_r}, \qquad (28)\end{aligned}$$

where Inequalities (26) and (27) follow from the binomial theorem.

First, consider the case where $|\Sigma|\ge 3$. In this case, one can verify that

$$\left(\sqrt[4]{8}+\sqrt[4]{2}\right)\left(1+\sqrt{|\Sigma|-1}\right)+|\Sigma|-2\le\sqrt{2}\left(|\Sigma|+2\sqrt{|\Sigma|-1}\right).$$

TABLE 2
Comparison of Bases of the Powers in the Time Bounds

	$\lvert\Sigma\rvert = 2$ (binary)	$\lvert\Sigma\rvert = 4$ (DNA)	$\lvert\Sigma\rvert = 20$ (protein)
$16(\lvert\Sigma\rvert - 1)$ [17]	16	48	304
$2^{3.25}(\lvert\Sigma\rvert - 1)$ [26]	9.51...	28.54...	180.75...
$\sqrt{2}\lvert\Sigma\rvert + \sqrt[4]{8}\left(\sqrt{2}+1\right)\left(1+\sqrt{\lvert\Sigma\rvert - 1}\right) - 2\sqrt{2}$ [this paper]		13.92...	47.21...
8 [this paper]	8		

Consequently, $T(d,d) \le (\sqrt{2}|\Sigma| + \sqrt[4]{8}(\sqrt{2}+1)(1+\sqrt{|\Sigma|-1}) - 2\sqrt{2})^d$ by Inequality (28).

Next, consider the case where $|\Sigma| = 2$. In this case, $\sum_{c=0}^{k-\ell_r}\binom{k}{c}(|\Sigma|-2)^c(\sqrt{2}(|\Sigma|+2\sqrt{|\Sigma|-1}))^m = (4\sqrt{2})^{m'}$, where $m' = \min\{d-k, k-\ell_r\}$. So, Inequality (25) can be simplified into the following:

$$T(d,d) \le \sum_{k=\ell_r}^{d} 2^{\frac{2d-k+\ell_r}{2}}\binom{d+\ell_r}{k}\left(4\sqrt{2}\right)^{m'}. \tag{29}$$

Note that $0.4m' \le 0.4d - 0.4k$ and $0.6m' \le 0.6k - 0.6\ell_r$. So, $m' \le 0.4d + 0.2k - 0.6\ell_r$. Plugging this inequality into Inequality (29), we have

$$\begin{aligned}T(d,d) &\le \sum_{k=\ell_r}^{d} 2^{\frac{2d-k+\ell_r}{2}}\binom{d+\ell_r}{k}\left(4\sqrt{2}\right)^{0.4d+0.2k-0.6\ell_r}\\ &= 2^{d+0.5\ell_r}\left(4\sqrt{2}\right)^{0.4d-0.6\ell_r}\sum_{k=\ell_r}^{d}\binom{d+\ell_r}{k}\left(\frac{(4\sqrt{2})^{0.2}}{\sqrt{2}}\right)^k\\ &= 2^{d+0.5\ell_r}\left(4\sqrt{2}\right)^{0.4d-0.6\ell_r}\sum_{k=\ell_r}^{d}\binom{d+\ell_r}{k}\\ &\le 2^{d+0.5\ell_r}\left(4\sqrt{2}\right)^{0.4d-0.6\ell_r}2^{d+\ell_r}\\ &= 8^d,\end{aligned}$$

where the last inequality follows from the binomial theorem. This completes the proof. □

Corollary 4.4. *If $|\Sigma| = 2$, then **CloseString2** solves the ECS problem in $O(nL + nd\cdot 8^d)$ time ; otherwise, it solves the problem in $O(nL + nd\cdot(\sqrt{2}|\Sigma| + \sqrt[4]{8}(\sqrt{2}+1)(1+\sqrt{|\Sigma|-1}) - 2\sqrt{2})^d)$ time.*

Proof. Similar to the proof of Corollary 3.5. □

The time bound for algorithm **CloseString2** may be difficult to appreciate. So, we compare it with the bounds of the algorithm in [17] and the best algorithm in [26]. Table 2 summarizes the result of comparison.

5 Extension to Closest Substring

In this section, we extend algorithm **CloseString** to an algorithm for the closest substring problem. We do not know if it is possible to extend algorithm **CloseString2** to an algorithm for the closest substring problem, because we do not know how to prove a lemma similar to Lemma 4.1 when the target problem becomes the closest substring problem.

Again, instead of solving the closest substring problem directly, we solve a more general problem called the *extended closest substring (ECSS) problem*. The input to the ECSS problem is a quintuple $(\mathcal{S}, d, t, P, b)$, where $\mathcal{S}$ is a set of strings of equal length (say, K), t is a string of some length L with $L \le K$, d is a positive integer less than or equal to L, P is a subset of $[1..L]$, and b is a nonnegative integer less than or equal to d. A *solution* to $(\mathcal{S}, d, t, P, b)$ is a string s' of length L satisfying the following conditions:

1. For every position $i \in P$, $s'[i] = t[i]$.
2. The number of positions i with $s'[i] \ne t[i]$ is at most b.
3. For every string $s \in \mathcal{S}$, s has a substring w of length L with $d(s', w) \le d$.

Given $(\mathcal{S}, d, t, P, b)$, the ECSS problem asks to output a solution if one exists. The output has to be a special symbol (say, Φ) if no solution exists.

Obviously, to solve the closest substring problem for a given instance $(\mathcal{S}, d, L)$, it suffices to perform the following three steps:

1. Select an arbitrary string $s \in \mathcal{S}$.
2. For every substring w of s with $|w| = L$, solve the ECSS problem for the instance $(\mathcal{S}, d, w, \emptyset, d)$.
3. If no solution is found in Step 2, output Φ and halt.

That is, we can solve the closest substring problem by calling any algorithm for the ECSS problem $K - L + 1$ times, where K is the length of a single input string. So, we hereafter focus on the ECSS problem instead of the closest substring problem, and design the following algorithm for solving it:

Algorithm 3: CloseSubstring

```
Input: An instance $(\mathcal{S}, d, t, P, b)$ of the ECSS problem.
Output: A solution to $(\mathcal{S}, d, t, P, b)$ if one exists, or $\Phi$ otherwise.
1. If every string $s \in \mathcal{S}$ has a substring $w$ with $|w| = |t|$ and
   $d(t, w) \le d$, then output $t$ and halt.
2. Find a string $s \in \mathcal{S}$ such that $s$ has no substring $w$ with $|w| = |t|$
   and $d(t, w) \le d$.
3. If all substrings $w$ of $s$ with $|w| = |t|$ satisfy
   $d(t, w) - d > \min\{b, |\{t \not\equiv w\} - P|\}$, then return $\Phi$.
4. For all substrings $w$ of $s$ with $|w| = |t|$ and
   $d(t, w) - d \le \min\{b, |\{t \not\equiv w\} - P|\}$, perform the following steps:
   4.1. Let $R = \{t \not\equiv w\} - P$ and $\ell = d(t, w) - d$.
   4.2. Make a copy $t'$ of $t$.
   4.3. For each subset $X$ of $R$ with $\ell \le |X| \le b$ and for each
        subset $Y$ of $X$ with $|Y| \le |X| - \ell$, perform the following
        steps:
        4.3.1. For each position $i \in X - Y$, change $t[i]$ to $w[i]$.
        4.3.2. For all $(|\Sigma| - 2)^{|Y|}$ possible ways to change the
               letters of $t$ at the positions in $Y$ (so that the letter of
               $t$ at each position $i \in Y$ is changed to a letter other
               than $w[i]$ and $t[i]$), change the letters of $t$ at the $|Y|$
               positions and then call CloseSubstring($\mathcal{S} - \{s\}$,
               $d, t, P \cup R, \min\{b - |X|, |X| - \ell - |Y|\}$) recursively.
        4.3.3. Restore $t$ back to $t'$.
5. Return $\Phi$.
```

Algorithm **CloseSubstring** is based on the following lemma which is similar to Lemma 3.1; its proof is omitted here because it is almost identical to that of Lemma 3.1.

Lemma 5.1. *Let $(\mathcal{S}, d, t, P, b)$ be an instance of the ECSS problem. Assume that s' is a solution to $(\mathcal{S}, d, t, P, b)$. Let s be a string in $\mathcal{S}$, let w be a substring of s with $|w| = |t|$ and $d(s', w) \le d$, let $\ell = d(t, w) - d$, let k be the number of positions $i \in \{t \not\equiv w\} - P$ with $s'[i] \ne t[i]$, let c be the number of positions $i \in \{t \not\equiv w\} - P$ with $s'[i] \ne t[i]$ and $s'[i] \ne w[i]$, and let b' be the number of positions $i \in [1..|t|] - (P \cup \{t \not\equiv w\})$ with $s'[i] \ne t[i]$. Then, $b' \le b - k$ and $b' \le k - \ell - c$. Consequently, $b' \le \frac{b-\ell-c}{2}$.*

Algorithm **CloseSubstring** is similar to algorithm **CloseString** in Section 3. The only difference is this: If the given input $(\mathcal{S}, d, t, P, b)$ has a solution s', then for each $s \in \mathcal{S}$, we need to guess a substring w of s with $|w| = |t|$ such that $d(w, s') \le b$. Note that s can have $|s| - |t| + 1$ substrings w with $|w| = |t|$.

Similar to the correctness of algorithm **CloseString**, we can prove the correctness of algorithm **CloseSubstring** based on Lemma 5.1. We next estimate its running time. Let $T'(d, b)$ denote the size of the search tree of algorithm **CloseSubstring** on input $(\mathcal{S}, d, t, P, b)$. Then, we have a lemma similar to Lemma 3.2:

Lemma 5.2. *Let K be the length of a single string in $\mathcal{S}$. Then,*

$$T'(d,b) \le \binom{d+b}{b} \cdot \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^b \cdot (K - |t| + 1)^{\lfloor \log_2(b+1) \rfloor}.$$

Proof. The proof is similar to that of Lemma 3.2 and hence is by induction on b. In case b is small enough that the algorithm does not make a recursive call, we have $T'(d, b) = 1$ and the lemma holds. So, assume that b is large enough that the algorithm makes at least one recursive call. For a string w of length $|t|$, let $\ell(w)$ denote $d(t, w) - d$. Then, by the algorithm, we have the following inequality:

$$T'(d,b) \le \sum_w \sum_{k=\ell(w)}^{b} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \, T'(d, \min\{b-k, k-\ell(w)-c\}), \tag{30}$$

where w ranges over all length-$|t|$ substrings of the string s selected in Step 2.

Let $\ell = \min_w \ell(w)$, where w ranges over all length-$|t|$ substrings of the string s selected in Step 2. For convenience, let $m = \min\{b-k, k-\ell-c\}$ and $h = K - L + 1$. Then, by Inequality (30) and the induction hypothesis,

$$\begin{aligned} T'(d,b) &\le h \sum_{k=\ell}^{b} \binom{d+b}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c h^{\lfloor \log_2(m+1) \rfloor} \binom{d+m}{m} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^m \\ &\le h^{\lfloor \log_2(b+1) \rfloor} \sum_{k=\ell}^{b} \binom{d+b}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \binom{d+m}{m} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^m, \end{aligned}$$

where the last inequality holds because $m \le \frac{b-\ell-c}{2} \le \frac{b-1}{2}$.

As in the proof of Lemma 3.2, we can prove the following inequality:

$$\sum_{k=\ell}^{b} \binom{d+b}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \binom{d+m}{m} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^m \le \binom{d+b}{b} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^b.$$

□

TABLE 3
Time Comparison of PMSPrune and Others in Challenging Cases of (L, d) [4]

Program	(11, 3)	(13, 4)	(15, 5)	(17, 6)	(19, 7)
PMSPrune	5s	53s	9m	69m	9.2h
PMSP	6.9s	152s	35m	12h	–
Voting	8.6s	108s	22m	–	–
RISOTTO	54s	600s	100m	12h	–

The following theorem shows that when $b = d$, we have a better bound on $T'(d, b)$ than the one in Lemma 5.2. Its proof is very similar to that of Theorem 3.4.

Theorem 5.3. $T'(d,d) \le (1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2})^d \cdot (K - |t| + 1)^{\lfloor \log_2(d+1) \rfloor}$.

Proof. If our algorithm makes no recursive calls, then the theorem clearly holds. So, assume that our algorithm makes at least one recursive call. Then, by Inequality (30), we have

$$T'(d,d) \le \sum_w \sum_{k=\ell(w)}^{d} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \, T'(d, \min\{d-k, k-\ell(w)-c\}).$$

Using this inequality, one can easily verify that the theorem holds when $d \le 2$. So, in the sequel, we assume that $d \ge 3$.

For convenience, let $b' = \min\{d-k, k-\ell(w)-c\}$ and $h = K - L + 1$. Note that $b' \le \frac{d-1-c}{2}$. By Lemma 5.2,

$$\begin{aligned} T'(d,d) &\le \sum_w \sum_{k=\ell(w)}^{d} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \binom{d+b'}{b'} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^{b'} h^{\lfloor \log_2(b'+1) \rfloor} \\ &\le h^{\lfloor \log_2(d+1) \rfloor - 1} \sum_w \sum_{k=\ell(w)}^{d} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \binom{d+b'}{b'} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^{b'}. \end{aligned}$$

As in the proof of Theorem 3.4, we can prove the following inequality:

$$\sum_{k=\ell(w)}^{d} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \binom{d+b'}{b'} \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^{b'} \le \left(1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2}\right)^d.$$

This finishes the proof. □

Corollary 5.4. *Let $m = K - L + 1$. Algorithm **CloseSubstring** solves the ECSS problem in time*

$$O\left((n-1)m\left(L + d \cdot \left(1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2}\right)^d \cdot m^{\lfloor \log_2(d+1) \rfloor}\right)\right).$$

TABLE 4
Time Comparison of Provable and PMSPrune in Practical Cases of (L, d)

Program	(10, 2)	(11, 2)	(12, 3)	(13, 3)	(14, 4)	(15, 4)	(16, 5)	(17, 5)	(18, 6)
Provable	0.20s	0.20s	0.81s	0.33s	15.98s	2.42s	4.55m	48.73s	59.30m
PMSPrune	0.16s	0.11s	1.36s	0.51s	20.25s	5.07s	3.97m	50.71s	34.83m

Consequently, the closest substring problem can be solved in time

$$O\left((n-1)m^2\left(L + d\cdot\left(1.5\sqrt[3]{2}|\Sigma| + 3\sqrt{3(|\Sigma|-1)} + 3\sqrt{3} - 3\sqrt[3]{2}\right)^d \cdot m^{\lfloor \log_2(d+1)\rfloor}\right)\right).$$

When implementing algorithm **CloseSubstring**, it is important to perform the following preprocessing:

- For each string $s \in \mathcal{S}$, compute the set $\mathcal{W}_s$ of all substrings w of s with $|w| = |t|$ and $d(t, w) \leq 2d$.

Suppose that we have done the above preprocessing. Then, when performing Steps 1 through 4 of the algorithm, we don't have to search for w in the whole s but rather only in $\mathcal{W}_s$. This simple idea was first used in [4] and can save us a lot of time.

6 Application to the Planted (L, d)-Motif Problem

An algorithm for the planted (L, d)-motif problem is *exact* if it finds all center substrings for each given input. We can transform our algorithm in Section 5 for the closest substring problem into an exact algorithm for the planted (L, d)-motif problem, by modifying Step 1 as follows:

1. If every string $s \in \mathcal{S}$ has a substring w with $|w| = |t|$ and $d(t, w) \leq d$, then return t.

In other words, our algorithm for the planted (L, d)-motif problem outputs all solutions instead of only one. Note that all the time bounds proved in Section 5 still hold even if we require that the algorithm finds all solutions instead of only one.

We have implemented our new exact algorithm for the planted (L, d)-motif problem in C. We call the resulting program *Provable* because its time complexity can be proved rigorously to be good, as already shown in Section 5. As in previous studies, we produce problem instances as follows: First, a motif consensus M of length L is chosen by picking L bases at random. Second, $n = 20$ occurrences of the motif are created by randomly choosing d positions per occurrence (without replacement) and mutating the base at each chosen position to a different, randomly chosen base. Third, we construct $n = 20$ background sequences of length $K = 600$ using $n \cdot K$ bases chosen at random. Finally, we assign each motif occurrence to a random position in a background sequence, one occurrence per sequence. All random choices are made uniformly and independently with equal base frequencies.

To show that Provable runs fast, we compare it against the previously fastest exact program (called PMSPrune [4]) for the planted (L, d)-motif problem. Table 3, copied from the paper [4], summarizes the average running times of PMSPrune and other previously known exact programs for some challenging cases of the planted (L, d)-motif problem.

Since PMSPrune is obviously the previously fastest, we ignore the other previously known exact programs and only run Provable and PMSPrune on the same randomly generated instances and counted their running times. Table 4 summarizes the average running times of Provable and PMSPrune on a 3.33 GHz Windows PC for practical problem instances, each randomly generated as described above. The number of tested instances for each case is 10. As can be seen from the table, our program runs faster for the cases where $(L, d) = (12, 3)$, $(13, 3)$, $(14, 4)$, $(15, 4)$, or $(17, 5)$.

Table 5 summarizes the average running times of Provable and PMSPrune on a 3.33 GHz Windows PC for challenging problem instances, each randomly generated as described above. Again, the number of tested instances for each case is 10. As can be seen from the table, our program runs faster for the cases where $(L, d) = (9, 2)$ or $(11, 3)$.

As can be seen from Tables 4 and 5, Provable runs slower than PMSPrune for some hard cases such as the cases where $(L, d) = (18, 6)$, $(17, 6)$, or $(19, 7)$.

Table 6 summarizes the behavior of Provable and that of PMSPrune when the motif length L changes. As can be seen from the table, both programs do not necessarily slow down when L increases; instead, they significantly slow down when (L, d) becomes a challenging instance.

In summary, it turns out that except really hard challenging cases of (L, d) (such as $(17, 6)$ and $(19, 7)$), our new program Provable runs well compared to (the buggy version of) PMSPrune.

7 Conclusion

We have presented a fast exact algorithm for the closest substring problem and have also implemented it (in C) into a program for the planted (L, d)-motif problem. The program is available upon request to the first author. We have run the program and compared it with the previously fastest program, namely, PMSPrune [4]. Our experimental data show that our program runs well compared to the buggy version of PMSPrune. It remains to be seen how much better it will run compared to the corrected version of PMSPrune.

Acknowledgments

The authors would like to thank three anonymous referees for very helpful comments. Zhi-Zhong Chen was supported in part by the

TABLE 5
Time Comparison of Provable and PMSPrune in Challenging Cases of (L, d)

Program	(9, 2)	(11, 3)	(13, 4)	(15, 5)	(17, 6)	(19, 7)
Provable	0.33s	6.09s	1.78m	25.97m	5.13h	49.85h
PMSPrune	0.58s	6.38s	1.02m	8.55m	1.23h	12.75h

TABLE 6
Time Comparison of Provable and PMSPrune for Different L

Program	(17, 6)	(18, 6)	(19, 6)	(20, 6)
Provable	5.13h	62.50m	11.58m	2.20m
PMSPrune	1.23h	34.83m	10.19m	2.41m

Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 20500021. Lusheng Wang was fully supported by a grant from City University of Hong Kong (Project No. 7002452).

REFERENCES

[1] A. Andoni, P. Indyk, and M. Patrascu, "On the Optimality of the Dimensionality Reduction Method," *Proc. 47th IEEE Symp. Foundations of Computer Science,* pp. 449-458, 2006.

[2] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi, "Banishing Bias from Consensus Sequences," *Proc. Eighth Symp. Combinatorial Pattern Matching,* pp. 247-261, 1997.

[3] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," *J. Computational Biology,* vol. 9, pp. 225-242, 2002.

[4] J. Davila, S. Balla, and S. Rajasekaran, "Fast and Practical Algorithms for Planted (l, d) Motif Search," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 4, no. 4, pp. 544-552, Oct.-Dec. 2007.

[5] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang, "Genetic Design of Drugs without Side-Effects," *SIAM J. Computing,* vol. 32, pp. 1073-1090, 2003.

[6] J. Dopazo, A. Rodríguez, J.C. Sáiz, and F. Sobrino, "Design of Primers for PCR Amplification of Highly Variable Genomes," *Computer Applications in the Biosciences,* vol. 9, pp. 123-125, 1993.

[7] R.G. Downey and M.R. Fellows, *Parameterized Complexity.* Springer, 1999.

[8] M.R. Fellows, J. Gramm, and R. Niedermeier, "On the Parameterized Intractability of Motif Search Problems," *Combinatorica,* vol. 26, pp. 141-167, 2006.

[9] M. Frances and A. Litman, "On Covering Problems of Codes," *Theoretical Computer Science,* vol. 30, pp. 113-119, 1997.

[10] J. Gramm, F. Huffner, and R. Niedermeier, "Closest Strings, Primer Design, and Motif Search," *Currents in Computational Molecular Biology,* poster abstracts of RECOMB 2002, pp. 74-75, 2002.

[11] J. Gramm, R. Niedermeier, and P. Rossmanith, "Fixed-Parameter Algorithms for Closest String and Related Problems," *Algorithmica,* vol. 37, pp. 25-42, 2003.

[12] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, "Distinguishing String Selection Problems," *Information and Computation,* vol. 185, pp. 41-55, 2003.

[13] M. Li, B. Ma, and L. Wang, "Finding Similar Regions in Many Sequences," *J. Computer and System Sciences,* vol. 65, pp. 73-96, 2002.

[14] M. Li, B. Ma, and L. Wang, "On the Closest String and Substring Problems," *J. ACM,* vol. 49, pp. 157-171, 2002.

[15] X. Liu, H. He, and O. Sýkora, "Parallel Genetic Algorithm and Parallel Simulated Annealing Algorithm for the Closest String Problem," *Proc. First Int'l Conf. Advanced Data Mining and Applications (ADMA '05),* pp. 591-597, 2005.

[16] K. Lucas, M. Busch, S. össinger, 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," *Computer Applications in the Biosciences,* vol. 7, pp. 525-529, 1991.

[17] B. Ma and X. Sun, "More Efficient Algorithms for Closest String and Substring Problems," *Proc. 12th Ann. Int'l Conf. Research in Computational Molecular Biology*, pp. 396-409, 2008.

[18] D. Marx, "The Closest Substring Problem with Small Distances," *Proc. 46th IEEE Symp. Foundations of Computer Science,* pp. 63-72, 2005.

[19] H. Mauch, M.J. Melzer, and J.S. Hu, "Genetic Algorithm Approach for the Closest String Problem," *Proc. Second IEEE Computer Soc. Bioinformatics Conf.,* pp. 560-561, 2003.

[20] C.N. Meneses, Z. Lu, C.A.S. Oliveira, and P.M. Pardalos, "Optimal Solutions for the Closest String Problem via Integer Programming," *INFORMS J. Computing,* vol. 16, pp. 419-429, 2004.

[21] P. Pevzner and S.-H. Sze, "Combinatorial Approaches to Finding Subtle Signals in DNA Sequences," *Proc. Eighth Int'l Conf. Intelligent Systems for Molecular Biology*, pp. 269-278, 2000.

[22] V. Proutski and E.C. Holme, "Primer Master: A New Program for the Design and Analysis of PCR Primers," *Computer Applications in the Biosciences,* vol. 12, pp. 253-255, 1996.

[23] N. Stojanovic, P. Berman, D. Gumucio, R. Hardison, and W. Miller, "A Linear-Time Algorithm for the 1-Mismatch Problem," *Proc. Fifth Int'l Workshop Algorithms and Data Structures,* pp. 126-135, 1997.

[24] Y. Wang, W. Chen, X. Li, and B. Cheng, "Degenerated Primer Design to Amplify the Heavy Chain Variable Region from Immunoglobulin cDNA," *BMC Bioinformatics,* vol. 7, suppl. 4, p. S9, 2006.

[25] L. Wang and L. Dong, "Randomized Algorithms for Motif Detection," *J. Bioinformatics and Computational Biology,* vol. 3, pp. 1038-1052, 2005.

[26] L. Wang and B. Zhu, "Efficient Algorithms for the Closest String and Distinguishing String Selection Problems," *Proc. Third Int'l Workshop Frontiers in Algorithms,* pp. 261-270, 2009.