A Fast Exact Algorithm for the Closest Substring Problem and Its Application to the Planted (L, d)-Motif Model

Zhi-Zhong Chen* Lusheng Wang †

Abstract

The closest substring problem has been studied extensively in the literature because of its wide applications in motif detection, universal PCR primer design, genetic drug target identification, binding sites locating, genetic probes design, etc. Since the problem is NP-hard, fixed-parameter algorithms and approximation algorithms have been designed for it. In this paper, we present a new fixed-parameter algorithm for the problem that is much faster than all existing algorithms. The running time of our algorithm depends on the alphabet size. The new algorithm runs in $O\left((n-1)m^2\left(L+d\cdot 17.97^d\cdot m^{\lfloor\log_2(d+1)\rfloor}\right)\right)$ time for DNA strings, and runs in $O\left((n-1)m^2\left(L+d\cdot 61.86^d\cdot m^{\lfloor\log_2(d+1)\rfloor}\right)\right)$ time for protein strings, where n is the number of input strings, L is the length of the output 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. In comparison, the previously best algorithm (due to Ma and Sun) for the closest substring problem runs in $O\left((n-1)m^2\left(L+d\cdot 48^d\cdot m^{\lfloor\log_2(d+1)\rfloor}\right)\right)$ time for DNA sequences, and runs in $O\left((n-1)m^2\left(L+d\cdot 304^d\cdot m^{\lfloor\log_2(d+1)\rfloor}\right)\right)$ time for protein sequences. The improvement is achieved by a new technique that makes use of different letters in the alphabet Σ, together with novel analysis of the running time. Due to its much faster speed, our new algorithm for the closest substring problem can be implemented into a C program that can give exact solutions for practical data sets of motif detection problems in a few seconds/minutes. To our knowledge, this is the first practical program that can give exact solutions and has a good theoretical bound on its running time for the NP-hard closest substring problem.

We apply our new exact program for the closest substring problem 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 which was designed and called PMSPrune by Davila, Balla, and Rajasekaran in 2007. Our experimental data show that our program runs faster 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 program 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 pointed out a bug in PMSPrune to its authors and they are still busy fixing it. PMSPrune is complicated and is hence bug-prone because it is based on branch-and-bound. Indeed, for certain problem instances, its current bugged version misses some solutions that should be output by the program. We believe that after the bug is fixed, PMSPrune will run significantly slower than its current bugged version. In addition, our algorithm uses less memory. To demonstrate the usefulness of exact algorithms for the problem, we also compare our program with a previous program called PROJECTION by Buhler and Tompa (which is based on random projection of the input strings). Our experimental data show that our program can achieve much better prediction accuracy in all tested cases (within almost the same running time and space).

Keywords: Fixed-parameter algorithm, closest string, closest substring, DNA motif discovery.

*Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan. Email: zzchen@mail.dendai.ac.jp

†Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. Email: cswangl@cityu.edu.hk

1 Introduction

Given a set $\mathcal{S}$ of n strings of equal length L and an integer d (called *radius*), the *closest string problem* asks to compute a center string s' (if exists) of length L such that the Hamming distance $d(s', s)$ between s' and each string $s \in \mathcal{S}$ is at most d. A more general problem is the *closest substring problem*. Given a set $\mathcal{S}$ of n strings of equal length K and two integers d and L, the closest substring problem asks to compute a center substring s' of length L such that each string $s \in \mathcal{S}$ has a substring t (if exists) of length L with $d(s', t) \leq d$. The two problems have been formulated and studied in a variety of applications in bioinformatics and computational biology, such as PCR primer design [6, 9, 11, 15, 21, 23], genetic probe design [11], antisense drug design [5, 11], finding unbiased consensus of a protein family [2], and motif finding [4, 7, 9, 11, 12, 24]. All these applications share a task that requires the design of a new DNA or protein sequence that is very similar to (a substring of) each of the given sequences.

Unfortunately, the closest string and substring problems have been proved to be NP-hard [8, 11]. This has motivated researchers to come up with heuristics without performance guarantee [14, 18, 19], fixed-parameter algorithms [10, 16, 22, 25], and polynomial-time approximation algorithms [1, 2, 5, 11, 13] for the problems. We here design fixed-parameter algorithms for the problems.

All known fixed-parameter algorithms for the closest string problem take d as the parameter and run in time exponential in d. Stojanovic *et al.* [22] presented a linear-time algorithm for the problem for $d = 1$ only. Gramm *et al.* [10] designed the first fixed-parameter algorithm for the problem. Their algorithm runs in $O\left(nL + nd \cdot (d+1)^d\right)$ time. Ma and Sun [16] improved Gramm *et al.*'s algorithm to obtain a new fixed-parameter algorithm for the problem that runs in $O\left(nL + nd \cdot (16(|\Sigma| - 1))^d\right)$ 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 give in [25], which runs in $O\left(nL + nd \cdot (2^{3.25}(|\Sigma| - 1))^d\right)$ time.

In this paper, we propose a new algorithm for the closest string problem whose time complexity is $O\left(nL + nd \cdot 47.21^d\right)$ for protein strings and is $O\left(nL + nd \cdot 13.92^d\right)$ for DNA strings. In comparison, the previous best algorithm in [25] runs in $O\left(nL + nd \cdot 180.75^d\right)$ time for protein strings and runs in $O\left(nL + nd \cdot 28.54^d\right)$ time for DNA strings. We achieve this improvement by a new technique that makes use of different letters in Σ. The new technique 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, previous algorithms do not look at the letters of s. In contrast, our algorithm looks at the letters of s and 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 simple but crucial lemma (Lemma 3.1) which says that the more such positions i exist, the faster our algorithm is. With this 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 $\binom{2d}{d}\left(|\Sigma| + 2\sqrt{|\Sigma| - 1}\right)^d$. This lemma is a significant improvement over the main theorem (Theorem 1) in [16], which roughly says that the total number of guesses made by the algorithm in [16] 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 [17] designed a fixed-parameter algorithm which runs in $O\left(|\Sigma|^{d(\log_2 d+2)} \cdot N^{\log_2 d+O(1)}\right)$ time, where N is the total length of input strings. Ma and Sun [16] obtained a faster algorithm running in $O\left(nm^2L + nd \cdot (16(|\Sigma|-1))^d \cdot m^{\lfloor \log_2(d+1) \rfloor+2}\right)$ time with $m = K-L+1$, by extending their fixed-parameter algorithm for the closest string problem. We can also extend one of our new fixed-parameter algorithms for the closest string problem to an algorithm for the closest substring problem that runs in $O\left((n-1)m^2\left(L + d \cdot 17.97^d \cdot m^{\lfloor \log_2(d+1) \rfloor}\right)\right)$ time for DNA strings and runs in $O\left((n-1)m^2\left(L + d \cdot 61.86^d \cdot m^{\lfloor \log_2(d+1) \rfloor}\right)\right)$ time for protein strings.

Due to its much faster speed, our new algorithm for the closest substring problem can be implemented into a C program that can give exact solutions for practical data sets of motif detection problems in a few seconds/minutes. To our knowledge, this is the first practical program that can give exact solutions and has a good theoretical bound on its running time for the NP-hard closest substring problem. We apply our program to the following problem proposed in [20]:

> **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 [20, 3], we fix $n = 20$ and $K = 600$. We compare our exact program with the previously best exact program called PMSPrune (due to Davila, Balla, and Rajasekaran [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 pointed out a bug in PMSPrune to its authors and they are still busy fixing it. PMSPrune is complicated and is hence bug-prone because it is based on branch-and-bound. Indeed, for certain problem instances, its current bugged version misses some solutions that should be output by the program. We believe that after the bug is fixed, PMSPrune will run significantly slower than its current bugged version. In contrast, our program is simple and is very likely to be bug-free. In addition, ours uses only $O(nm)$ space while PMSPrune needs $O(nm^2)$ space.

Our program can be applied to real biological data as follows. First, use it to output all solutions (one of which must be correct). Then, use a known scoring scheme to evaluate the solutions so that the correct one gets the highest score. For example, for the planted (L, d)-motif problem, we know that the correct solution (i.e., the planted motif consensus) has a Hamming distance of d from some substring of each input string. Thus, we can give the highest score to each solution that has a Hamming distance of d from some substring of each input string. We combine this kind of scoring scheme with our program and compare the resulting program with a previous program called PROJECTION by Buhler and Tompa [3] which is based on random projection of the input strings. Our experimental data show that our program almost always recovers the correct solution and achieves a much better average performance coefficient even in challenging cases such as the cases where $(L, d) = (9, 2)$, $(11, 3)$, or $(15, 5)$. Similar to PROJECTION, our program typically

solves practical cases in only a few seconds/minutes and solves very hard cases (such as the cases where $(L, d) = (18, 6)$ or $(17, 6)$) in one or a few hours.

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.

Due to lack of space, the proofs of all lemmas and theorems have been moved to the appendix.

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, ..., $|s|$. 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 $s[i]$. For two strings s and t of equal length, $P(s, t)$ denotes the set of positions $i \in [1..|s|]$ with $s[i] \neq t[i]$, while $d(s, t)$ denotes $|P(s, t)|$ (i.e., the Hamming distance between s and 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, NULL) 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 all previous algorithms: 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 letters of t should be modified. For each guessed position i, all previous algorithms 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 $i \in P(t,s) - P$ with $s'[i] \neq t[i]$, let c be the number of positions $i \in P(t,s) - P$ with $s'[i] \neq t[i]$ and $s'[i] \neq s[i]$, and let b' be the number of positions $i \in [1..L] - (P \cup P(t,s))$ with $s'[i] \neq t[i]$. Then, $b' \leq b - k$ and $b' \leq k - \ell - c$. Consequently, $b' \leq \frac{b-\ell-c}{2}$.*

We note that Lemma 3.1 is stronger than Lemma 1 in [16] and Lemma 2 in [25].

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 NULL 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 $R = P(t,s) - P$.

4. If $\ell > \min\{b, |R|\}$, then return NULL.

5. 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:

  5.1. For each position $i \in X - Y$, change $t[i]$ to $s[i]$.

  5.2. 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. Output NULL and halt.

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 $d(t,s) = |P(t,s)| = d + \ell$. So, in order to satisfy $d(t,s) \leq d$, we need to first select at least ℓ positions among the positions in $P(t,s)$ 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 $R = P(t,s) - P$. So, no solution exists if $\ell > \min\{b, |R|\}$. The correctness of Step 5.2 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 $P \cup P(t,s) = P \cup R$) of t where the letters have to be modified in order to transform t into a solution.

The remainder of this section is devoted to estimating the running time of the algorithm. The recursive calls made by algorithm **CloseString** can be organized into a tree in which each 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 this tree the *search tree* on input $(\mathcal{S}, d, t, P, b)$. The following lemma shows an upper bound on the size of the search tree.

Lemma 3.2 *Let $T(d,b)$ be the number of nodes in the search tree on input $(\mathcal{S}, d, t, P, b)$. Then, $T(d,b) \leq \binom{d+b}{b}(|\Sigma| - 1)^b \; 2^{2\nu b}$, where $\nu = \log_2 \frac{1+\sqrt{|\Sigma|-1}}{\sqrt{|\Sigma|-1}}$.*

We note that Lemma 3.2 is much stronger than Theorem 1 in [16], which seems to be the main result in [16]. Indeed, the proof of Lemma 3.2 also looks simpler than that of Theorem 1 in [16].

By Lemma 3.2, $T(d,d)$ can be bounded from above roughly by $\binom{2d}{d}(|\Sigma|-1)^d$ if $|\Sigma|$ is large enough. In particular, our upper bound $\binom{2d}{d}(|\Sigma|-1)^d\ 2^{2\nu d}$ is already better than the upper bound $\binom{2d}{d}(|\Sigma|-1)^d\ 2^{1.25d}$ in [25], if $|\Sigma| \geq 5$.

We next show an even better upper bound on $T(d,d)$. The following lemma will be very helpful:

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

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

Theorem 3.4 *Let $\mu = 1 + \frac{\sqrt[3]{2}(|\Sigma|-2)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}$. Then, $T(d,d) \leq \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2} \cdot (\mu+1)\right)^d$.*

Corollary 3.5 *Let μ be as in Theorem 3.4. Algorithm* **CloseString** *solves the ECS problem in $O\left(nL + nd \cdot \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2} \cdot (\mu+1)\right)^d\right)$ time.*

The better algorithm in [25] for the closest string problem runs in $O\left(nL + nd \cdot \left(2^{3.25}(|\Sigma|-1)\right)^d\right)$ time, while the algorithm in [16] for the same problem runs in $O\left(nL + nd \cdot \left(16(|\Sigma|-1)\right)^d\right)$ 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.

Table 1: Comparison of bases of the powers in the time bounds.

	$\|\Sigma\| = 4$ (DNA)	$\|\Sigma\| = 20$ (protein)
$16(\|\Sigma\|-1)$ [16]	48	304
$2^{3.25}(\|\Sigma\|-1)$ [25]	28.54...	180.75...
$\frac{\sqrt{27}+\sqrt{27(\|\Sigma\|-1)}}{2} \cdot (\mu+1)$ [this paper]	17.97...	61.86...

4 The Second Algorithm

Motivated by an idea in [25], 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

Table 2: Comparison of bases of the powers in the time bounds.

	$\|\Sigma\| = 2$ (binary)	$\|\Sigma\| = 4$ (DNA)	$\|\Sigma\| = 20$ (protein)
$16(\|\Sigma\| - 1)$ [16]	16	48	304
$2^{3.25}(\|\Sigma\| - 1)$ [25]	9.51...	28.54...	180.75...
$\sqrt[4]{8}\left(1 + \sqrt{\|\Sigma\| - 1}\right)\left(\sqrt{2} + \gamma\right)$ [this paper]		13.92...	47.21...
$4 + 3\sqrt{2}$ [this paper]	8.24 ...		

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 Step 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 5 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, $|P_u \cap P(t_u, s_u)| \geq \frac{d(t_u,t_r)-\ell_r+\ell_u}{2}$.*

We note in passing that there is a lemma in [25] 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 $\mathcal{T}$, $F(d, b_u) \leq \binom{\lfloor \frac{2d-d(t_u,t_r)+\ell_r+b_u}{2} \rfloor}{b_u}(|\Sigma| - 1)^{b_u} 2^{2\nu b_u}$.*

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

Theorem 4.3 *Let $\gamma = 1 + \frac{|\Sigma|-2}{\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)}$. If $|\Sigma| = 2$, then the size $T(d, d)$ of tree $\mathcal{T}$ is at most $(4 + 3\sqrt{2})^d$. Otherwise, $T(d, d) \leq \left(\sqrt[4]{8}\left(1 + \sqrt{|\Sigma| - 1}\right)\left(\sqrt{2} + \gamma\right)\right)^d$.*

Corollary 4.4 *Let γ be as in Theorem 4.3. If $|\Sigma| = 2$, then algorithm* **CloseString2** *solves the ECS problem in time $O\left(nL + nd \cdot \left(4 + 3\sqrt{2}\right)^d\right)$; otherwise, it solves the problem in time $O\left(nL + nd \cdot \left(\sqrt[4]{8}\left(1 + \sqrt{|\Sigma| - 1}\right)\left(\sqrt{2} + \gamma\right)\right)^d\right)$.*

The time bound for algorithm **CloseString2** may be difficult to appreciate. So, we compare it with the bounds of the algorithm in [16] and the faster algorithm in [25]. 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 $(\mathcal{S}, t, d, P, b)$, where $\mathcal{S}$ is a set of strings of equal length (say, K), t is a string of some length L with $L \leq K$, 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 $(\mathcal{S}, t, d, P, b)$ 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 $s'[i] \neq t[i]$ is at most b.
3. For every string $s \in \mathcal{S}$, s has a substring w of length L with $d(s', w) \leq d$.

Given $(\mathcal{S}, t, d, P, b)$, the ECSS problem asks to output a solution if one exists. The output has to be a special symbol (say, NULL) 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 $(\mathcal{S}, w, d, \emptyset, d)$.
3. If no solution is found in Step 2, output NULL and halt.

That is, we can solve the closest substring problem by calling any algorithm for the ECSS problem $K - L + 1$ times, where L 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 $(\mathcal{S}, t, d, P, b)$ of the ECSS problem.

Output: A solution to $(\mathcal{S}, t, d, P, b)$ if one exists, or NULL otherwise.

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

2. Find a string $s \in \mathcal{S}$ such that s has no substring w with $|w| = |t|$ and $d(t, w) \leq d$.

3. If all substrings w of s with $|w| = |t|$ satisfy $d(t, w) - d > \min\{b, |P(t, w) - P|\}$, then return NULL.

4. For all substrings w of s with $|w| = |t|$ and $d(t, w) - d \leq \min\{b, |P(t, w) - P|\}$, perform the following steps:

4.1. Let $R = P(t, w) - P$ and $\ell = d(t, w) - d$.

4.2. 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:

4.2.1. For each position $i \in X - Y$, change $t[i]$ to $w[i]$.

4.2.2. 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**$(\mathcal{S} - \{s\}, t, d, P \cup R, \min\{b - |X|, |X| - \ell - |Y|\})$ recursively.

5. Output NULL and halt.

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

Algorithm **CloseSubstring** is similar to algorithm **CloseString** in Section 3. The only difference is this: If the given input $(\mathcal{S}, t, d, P, b)$ 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 $d(w, s') \leq b$. Note that s can have $|s| - |t| + 1$ substrings with $|w| = |t|$.

As we proved 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 $(\mathcal{S}, t, d, P, b)$. Then, we have a lemma similar to Lemma 3.2:

Lemma 5.2 *Let ν be as in Lemma 3.2, and let K be the length of a single string in $\mathcal{S}$. Then, $T'(d, b) \leq \binom{d+b}{b} \cdot (|\Sigma| - 1)^b \cdot 2^{2\nu b} \cdot (K - |t| + 1)^{\lfloor \log_2(b+1) \rfloor}$.*

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 $T'(d, d) \leq \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2} \cdot (\mu + 1) \right)^d \cdot (K - |t| + 1)^{\lfloor \log_2(d+1) \rfloor}$, *where μ is as in Theorem 3.4.*

Corollary 5.4 *Let $m = K - L + 1$. Algorithm* **CloseSubstring** *solves the ECSS problem in time* $O\left((n-1)m^2 \left(L + d \cdot \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2} \cdot (\mu + 1) \right)^d \cdot m^{\lfloor \log_2(d+1) \rfloor} \right) \right)$.

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

A program for the planted (L, d)-motif problem is *exact* if it finds all center substrings for each given input. Since one of the center substrings must be the planted motif consensus, we can use a known scoring scheme to evaluate the center substrings so that the planted motif consensus gets the highest score.

We have implemented our algorithm in Section 5 for the closest substring problem into an exact program (in C) for the planted (L, d)-motif problem. We call the program *Provable* because its time comlexity 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

Table 3: Performance Comparison of PROJECTION and Provable.

	PROJECTION		Provable	
(L, d)	a.p.c.	Correct	a.p.c.	Correct
$(10, 2)$	0.80	100/100	0.88	100/100
$(11, 2)$	0.94	100/100	0.96	100/100
$(12, 3)$	0.77	96/100	0.93	100/100
$(13, 3)$	0.94	100/100	0.97	100/100
$(14, 4)$	0.71	86/100	0.86	100/100
$(15, 4)$	0.93	100/100	0.93	100/100
$(16, 5)$	0.67	77/100	0.88	100/100
$(17, 5)$	0.94	98/100	0.96	100/100
$(18, 6)$	0.73	82/100	0.90	100/100
$(19, 6)$	0.94	98/100	0.98	100/100
$(9, 2)$	0.28	11/20	0.55	90/100
$(11, 3)$	0.02	1/20	0.61	93/100
$(13, 4)$	0.06	2/20	0.57	90/100
$(15, 5)$	0.02	0/20	0.68	95/100

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.

For each randomly generated instance, we first run Provable to obtain all center substrings. To evaluate the center substrings, we use an obvious scoring scheme that favors those center substrings which have a distance of d to at least one substring of each input string. If two or more center substrings get the highest score, we select one of them randomly as the candidate motif consensus. Table 3 summarizes the experimental results together with a performance comparison to a previous program called PROJECTION by Buhler and Tompa [3].

In Table 3, the column titled "a.p.c." stands for the average performance coefficient (defined by Pevzner and Sze [20]) over 100 random problem instances and the column titled "Correct" shows the number of instances (out of 100) yielding correct motif consensus. Moreover, the data about PROJECTION are copied from the paper [3]. As can be seen from Table 3, for the tested cases, our algorithm can almost always find the planted motif consensus M and predict better occurrences of M in the input strings than PROJECTION.

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 4, 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 5 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. As can be seen from the table, our program runs faster for the cases where

Table 4: [4] Time Comparison of PMSPrune and others in Challenging Cases of (L, d).

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	–

$(L, d) = (12, 3)$, $(13, 3)$, $(14, 4)$, $(15, 4)$, or $(17, 5)$.

Table 5: 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

Table 6 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. As can be seen from the table, our program runs faster for the cases where $(L, d) = (9, 2)$ or $(11, 3)$.

Table 6: 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	107.08s	25.97m	5.13h	49.85h
PMSPrune	0.58s	6.38s	61.39s	8.55m	1.23h	12.75h

As can be seen from Tables 5 and 6, Provable runs slower than PMSPrune for some hard cases such as the cases where $(L, d) = (18, 6)$, $(17, 6)$, or $(19, 7)$. It should be pointed out that we have pointed out a bug in PMSPrune to its authors and they are still busy fixing it. PMSPrune is complicated and is hence bug-prone because it is based on branch-and-bound. Indeed, for certain problem instances, its current bugged version misses some solutions that should be output by the program. We believe that after the bug is fixed, PMSPrune will run significantly slower than its current bugged version. In contrast, our program is simple and is very likely to be bug-free.

Table 7 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.

Table 7: 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

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 bugged version of) PMSPrune.

References

[1] A. Andoni, P. Indyk, and M. Patrascu. On the Optimality of the Dimensionality reduction Method. *Proceedings of 47th IEEE Symposium on Foundations of Computer Science*, pp. 449-458, 2006.

[2] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi. Banishing Bias from Consensus Sequences. *Proceedings of 8th Symposium on Combinatorial Pattern Matching*, Lecture in Computer Science, **1264** (1997) 247-261.

[3] J. Buhler and M. Tompa. Finding Motifs Using Random Projections. *Journal of Computational Biology*, **9** (2002) 225-242.

[4] J. Davila, S. Balla, and S. Rajasekaran. Fast and Practical Algorithms for Planted (l, d) Motif Search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **4** (2007) 544-552.

[5] X. Deng, G. Li, Z. Li, B. Ma, and L. Wang. Genetic Design of Drugs without Side-Effects. *SIAM Journal on Computing*, **32** (2003) 1073-1090.

[6] J. Dopazo, A. Rodríguez, J. C. Sáiz, and F. Sobrino. Design of Primers for PCR Amplification of Highly Variable Genomes. *CABIOS*, **9** (1993) 123-125.

[7] M. R. Fellows, J. Gramm, and R. Niedermeier. On the Parameterized Intractability of Motif Search Problems. *Combinatorica*, **26** (2006) 141-167.

[8] M. Frances and A. Litman. On Covering Problems of Codes. *Theoretical Computer Science*, **30** (1997) 113-119.

[9] 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.

[10] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-Parameter Algorithms for Closest String and Related Problems. *Algorithmica*, **37** (2003) 25-42.

[11] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing String Selection Problems. *Information and Computation*, **185** (2003) 41-55. A preliminary version appeared in *Proceedings of 10th ACM-SIAM Symposium on Discrete Algorithms*, pp. 633-642, 2003.

[12] M. Li, B. Ma, and L. Wang. Finding Similar Regions in Many Sequences. *J. Comput. Syst. Sci.*, **65** (2002) 111-132. A preliminary version appeared in *Proceedings of the 31st ACM Symposium on Theory of Computing*, pp. 473-482, 1999.

[13] M. Li, B. Ma, and L. Wang. On the Closest String and Substring Problems. *J. ACM*, **49** (2002) 157-171.

[14] X. Liu, H. He, and O. Sýkora. Parallel Genetic Algorithm and Parallel Simulated Annealing Algorithm for the Closest String Problem. *Proceedings of ADMA 2005*, Lecture Notes in Computer Science, **3584** (2005) 591-597.

[15] 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. *CABIOS*, **7** (1991) 525-529.

[16] B. Ma and X. Sun. More Efficient Algorithms for Closest String and Substring Problems. *Proceedings of 12th International Conference on Research in Computational Molecular Biology*, Lecture Notes in Computer Science, **4955** (2008) 396-409. Also to appear in *SIAM Journal on Computing*.

[17] D. Marx. The Closest Substring Problem with Small Distances. *Proceedings of 46th IEEE Symposium on Foundations of Computer Science*, pp. 63-72, 2005.

[18] H. Mauch, M. J. Melzer, and J. S. Hu. Genetic Algorithm Approach for the Closest String Problem. *Proceedings of the 2nd IEEE Computer Society Bioinformatics Conference*, pp. 560-561, 2003.

[19] 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.

[20] P. Pevzner and S.-H. Sze. Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. *Proceedings of 8th Int. Conf. Intelligent Systems for Molecular Biology*, pp. 269-278, 2000.

[21] V. Proutski and E. C. Holme. Primer Master: A New Program for the Design and Analysis of PCR Primers. *CABIOS*, **12** (1996) 253-255.

[22] N. Stojanovic, P. Berman, D. Gumucio, R. Hardison, and W. Miller. A Linear-Time Algorithm for the 1-Mismatch Problem. *Proceedings of 5th International Workshop on Algorithms and Data Structures*, pp. 126-135, 1997.

[23] 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).

[24] L. Wang and L. Dong. Randomized Algorithms for Motif Detection. *Journal of Bioinformatics and Computational Biology*, **3** (2005) 1038-1052.

[25] L. Wang and B. Zhu. Efficient Algorithms for the Closest String and Distinguishing String Selection Problems. *Proceedings of 3rd International Workshop on Frontiers in Algorithms*, Lecture Notes in Computer Science, **5598** (2009) 261-270.

Appendix: Omitted Proofs

Proof of Lemma 3.1: (See Figure 1.) 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 $a = |P \cap P(t, s)|$, and let h be the number of positions $i \in P(t, s) - P$ with $t[i] = s'[i]$ (see Figure 1). Then, $|P(t, s)| = a+h+k$ and $d(s', s) = a+h+c+b'$. So, by assumption, $a+h+k = d+\ell$ and $a + h + c + b' \leq d$. Thus, $b' \leq k - \ell - c$. □

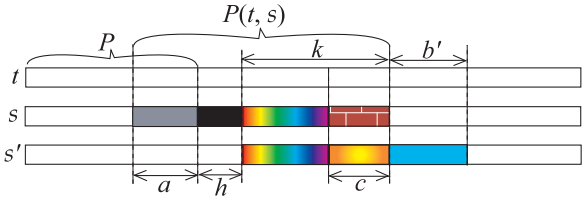

Figure 1: Strings t, s, and s' in Lemma 3.1, where for each position $i \in [1..L]$, $t[i] = s'[i]$ (respectively, $t[i] = s[i]$ or $s'[i] = s[i]$) if and only if $t[i]$ and $s'[i]$ (respectively, $t[i]$ and $s[i]$, or $s'[i]$ and $s[i]$) are illustrated in the same color.

Proof of Lemma 3.2: By induction on b. In case $b = 0$, the algorithm will output either t or NULL 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:

$$T(d,b) \leq \sum_{k=\ell}^{b} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \; T(d, \min\{b-k, k-\ell-c\}). \tag{6.1}$$

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

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

Since $m \leq b - k$, $m \leq \frac{b-\ell-c}{2}$, and $d \geq b$,

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

So, to finish the proof, it suffices to show that $\left(2+\frac{|\Sigma|-2}{\sqrt{|\Sigma|-1}\cdot 2^{\nu}}\right)^{b} \leq (|\Sigma|-1)^{\frac{b}{2}}\ 2^{\nu b}$, or equivalently $2+\frac{|\Sigma|-2}{\sqrt{|\Sigma|-1}\cdot 2^{\nu}} \leq 2^{\nu}\sqrt{|\Sigma|-1}$. The last inequality is further equivalent to $\left(2^{\nu}-\frac{1}{\sqrt{|\Sigma|-1}}\right)^{2} \geq 1$. This inequality holds because $\nu = \log_2 \frac{1+\sqrt{|\Sigma|-1}}{\sqrt{|\Sigma|-1}}$. □

Proof of Lemma 3.3: By Stirling's formula, for any integer $h \geq 0$, $h! = \sqrt{2\pi h}\left(\frac{h}{e}\right)^{h} e^{\lambda_h}$ with $\frac{1}{12h+1} < \lambda_h < \frac{1}{12h}$, where π is the ratio of the circumference of a circle to its diameter and e is the base of the natural logarithm. So,

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

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

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

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

□

Proof of Theorem 3.4: 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 6.1, we have

$$T(d,d) \leq \sum_{k=\ell}^{d}\binom{d+\ell}{k}\sum_{c=0}^{k-\ell}\binom{k}{c}(|\Sigma|-2)^{c}\ T(d,\min\{d-k,k-\ell-c\}). \tag{6.2}$$

Using Inequality 6.2 and the fact that $T(d,0) \leq 1$, one can easily verify that $T(1,1) \leq 2$ and $T(2,2) \leq 6|\Sigma|-6$. Since $2 \leq \frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2}\cdot(\mu+1)$ and $6|\Sigma|-6 \leq \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2}\cdot(\mu+1)\right)^{2}$, the theorem holds when $d \leq 2$. So, in the sequel, we assume that $d \geq 3$.

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

$$T(d,d) \leq \sum_{k=\ell}^{d}\binom{d+\ell}{k}\sum_{c=0}^{k-\ell}\binom{k}{c}(|\Sigma|-2)^{c}\ \binom{d+b'}{b'}\ (|\Sigma|-1)^{b'}\ 2^{2\nu b'}.$$

Since $b' \leq \frac{d-\ell-c}{2}$, $b' \leq \lfloor \frac{d+b'}{3} \rfloor$. Moreover, since $d \geq 3$, $1 \leq \lfloor \frac{d+b'}{3} \rfloor \leq d+b'-1$. So, by Lemma 3.3,

$$T(d,d) \leq \sum_{k=\ell}^{d}\binom{d+\ell}{k}\sum_{c=0}^{k-\ell}\binom{k}{c}(|\Sigma|-2)^{c}\ \left(\frac{3}{\sqrt[3]{4}}\right)^{d+b'}\ (|\Sigma|-1)^{b'}\ 2^{2\nu b'}.$$

Moreover, we clearly have $b' \leq \frac{d-\ell-c}{2}$ and $2^\nu = \frac{1+\sqrt{|\Sigma|-1}}{\sqrt{|\Sigma|-1}}$. Thus,

$$\begin{aligned}
T(d,d) &\leq \sum_{k=\ell}^{d} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} (|\Sigma|-2)^c \left(\frac{3}{\sqrt[3]{4}}\right)^{d+\frac{d-\ell-c}{2}} \left(2^\nu \sqrt{|\Sigma|-1}\right)^{d-\ell-c} \\
&= \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell} \left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell} \sum_{k=\ell}^{d} \binom{d+\ell}{k} \sum_{c=0}^{k-\ell} \binom{k}{c} \left(\frac{\sqrt[3]{2}(|\Sigma|-2)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}\right)^c. \\
&\leq \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell} \left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell} \sum_{k=\ell}^{d} \binom{d+\ell}{k} \left(1+\frac{\sqrt[3]{2}(|\Sigma|-2)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}\right)^k. \\
&= \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell} \left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell} \sum_{k=\ell}^{d} \binom{d+\ell}{k} \mu^k \\
&\leq \left(\frac{\sqrt{3}}{\sqrt[3]{2}}\right)^{3d-\ell} \left(1+\sqrt{|\Sigma|-1}\right)^{d-\ell} (1+\mu)^{d+\ell} \\
&\leq \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2}\right)^{d} (\mu+1)^d \left(\frac{\sqrt[3]{2}(\mu+1)}{\sqrt{3}+\sqrt{3(|\Sigma|-1)}}\right)^{\ell} \\
&\leq \left(\frac{\sqrt{27}+\sqrt{27(|\Sigma|-1)}}{2} \cdot (\mu+1)\right)^{d},
\end{aligned}$$

where the last inequality holds because $\sqrt[3]{2}(\mu+1) \leq \sqrt{3}+\sqrt{3(|\Sigma|-1)}$ for $|\Sigma| \geq 2$. This finishes the proof. □

Proof of Corollary 3.5: Obviously, each leaf of the search tree takes $O(nL)$ time. As observed in previous works (e.g., [16]), 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. □

Proof of Lemma 4.1: For convenience, we use $Q(w_1, w_2)$ to denote the set of positions $i \in [1..|w_1|]$ with $w_1[i] = w_2[i]$ for two strings w_1 and w_2 of equal length. Moreover, for ease of explanation, we define the following notations (cf. Figure 2):

- $i = |P(t_r, s_r) \cap P(t_r, t_u) \cap Q(t_r, s_u)|$.
- $j = |P(t_r, s_r) \cap P(t_r, t_u) \cap P(t_r, s_u)|$.
- $j' = |P(t_r, s_r) \cap P(t_r, t_u) \cap P(t_r, s_u) \cap P(t_u, s_u)|$.
- $h = |P(t_r, s_r) \cap Q(t_r, t_u) \cap P(t_r, s_u)|$.
- $g = |P(t_r, s_r) \cap Q(t_r, t_u) \cap Q(t_r, s_u)|$.
- $x = |\,(P_u - P(t_r, s_r)) \cap Q(t_r, t_u) \cap P(t_r, s_u)|$.
- $y = |\,(P_u - P(t_r, s_r)) \cap P(t_r, t_u) \cap P(t_r, s_u)|$.
- $y' = |\,(P_u - P(t_r, s_r)) \cap P(t_r, t_u) \cap P(t_r, s_u) \cap P(t_u, s_u)|$.

- $z = |\,(P_u - P(t_r, s_r)) \cap P(t_r, t_u) \cap Q(t_r, s_u)|$.
- $a = |\,(P(t_u, s_u) - P_u)\,|$, or equivalently $a = |\,(P(t_r, s_u) - P_u)\,|$.

By the above definitions, we have the following five equations immediately:

$$\begin{aligned} d(t_r, s_r) &= i + j + h + g = d + \ell_r && (6.3)\\ d(t_r, s_u) &= j + h + x + y + a && (6.4)\\ d(t_u, s_u) &= i + j' + h + x + y' + z + a = d + \ell_u && (6.5)\\ d(t_r, t_u) &= i + j + y + z && (6.6)\\ |P_u \cap P(t_u, s_u)| &= i + j' + h + x + y' + z. && (6.7) \end{aligned}$$

By Step 2 of algorithm **CloseString2**, $d(t_r, s_u) \leq d(t_r, s_r)$. So, Equations 6.3 and 6.4 imply the following inequality:

$$x + y + a \leq i + g. \tag{6.8}$$

By Equations 6.6 and 6.7, in order to finish the proof, we need only to prove the following inequality:

$$i + j' + h + x + y' + z \geq \frac{i + j - \ell_r + y + z + \ell_u}{2}. \tag{6.9}$$

By Equation 6.3, Inequality 6.9 is equivalent to the following inequality:

$$2i + 2j' + 3h + 2x + 2y' + z + g \geq d + \ell_u + y. \tag{6.10}$$

By Equation 6.5, Inequality 6.10 is equivalent to the following inequality:

$$i + j' + 2h + x + y' + g \geq a + y. \tag{6.11}$$

By Inequality 6.8, Inequality 6.11 holds. This finishes the proof. □

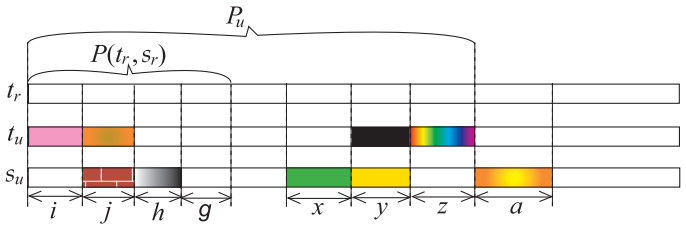

Figure 2: Strings t_r, t_u, and s_u in Lemma 4.1, where for each position $i \in [1..L]$, (1) $t_r[i] = t_u[i]$ (respectively, $t_r[i] = s_u[i]$ or $t_u[i] = s_u[i]$) if $t_r[i]$ and $t_u[i]$ (respectively, $t_r[i]$ and $s_u[i]$, or $t_u[i]$ and $s_u[i]$) are illustrated in the same color, and (2) $t_r[i] \neq t_u[i]$ (respectively, $t_r[i] \neq s_u[i]$ or $t_u[i] \neq s_u[i]$) if one of $t_r[i]$ and $t_u[i]$ (respectively, $t_r[i]$ and $s_u[i]$, or $t_u[i]$ and $s_u[i]$) is colored white but the other is not colored white.

Proof of Lemma 4.2: 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,

$$\begin{aligned} F(d,b_u) &\leq \sum_{k=\ell_u}^{b_u} \binom{d+\ell_u-\lceil\frac{d(t_u,t_r)-\ell_r+\ell_u}{2}\rceil}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma|-2)^c F(d, \min\{b_u-k, k-\ell_u-c\}) \\ &= \sum_{k=\ell_u}^{b_u} \binom{\lfloor\frac{2d-d(t_u,t_r)+\ell_r+\ell_u}{2}\rfloor}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma|-2)^c F(d, \min\{b_u-k, k-\ell_u-c\}). \quad (6.12) \end{aligned}$$

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 $\ell_u \leq b_u$ 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 6.12 and the induction hypothesis,

$$F(d,b_u) \leq \sum_{k=\ell_u}^{b_u} \binom{h}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma|-2)^c \binom{\lfloor\frac{2d-d(t_u,t_r)-k+\ell_r+m}{2}\rfloor}{m} (|\Sigma|-1)^m \ 2^{2\nu m}. \quad (6.13)$$

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

$$\begin{aligned} F(d,b_u) &\leq \sum_{k=\ell_u}^{b_u} \binom{h}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma|-2)^c \binom{\lfloor\frac{2d-d(t_u,t_r)-2k+\ell_r+b_u}{2}\rfloor}{b_u-k} (|\Sigma|-1)^m \ 2^{2\nu m} \\ &= \sum_{k=\ell_u}^{b_u} \binom{h}{k} \sum_{c=0}^{k-\ell_u} \binom{k}{c} (|\Sigma|-2)^c \binom{h-k}{b_u-k} (|\Sigma|-1)^m \ 2^{2\nu m}. \quad (6.14) \end{aligned}$$

Since $m \leq \frac{b_u-\ell_u-c}{2}$, Inequality 6.14 implies the following:

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

where the last inequality follows from the definition of ν. This finishes the proof. □

Proof of Theorem 4.3: 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

$$T(d,d) \leq \sum_{k=\ell_r}^{d} \binom{d+\ell_r}{k} \sum_{c=0}^{k-\ell_r} \binom{k}{c} (|\Sigma|-2)^c \ F(d, \min\{d-k, k-\ell_r-c\}). \quad (6.15)$$

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

$$\begin{aligned}
T(d,d) &\leq \sum_{k=\ell_r}^{d} \binom{d+\ell_r}{k} \sum_{c=0}^{k-\ell_r} \binom{k}{c} (|\Sigma|-2)^c \binom{\lfloor \frac{2d-k+\ell_r+m}{2} \rfloor}{m} (|\Sigma|-1)^m 2^{2\nu m} \\
&\leq \sum_{k=\ell_r}^{d} \binom{d+\ell_r}{k} \sum_{c=0}^{k-\ell_r} \binom{k}{c} (|\Sigma|-2)^c \, 2^{\frac{2d-k+\ell_r+m}{2}} (|\Sigma|-1)^m 2^{2\nu m} \\
&= \sum_{k=\ell_r}^{d} 2^{\frac{2d-k+\ell_r}{2}} \binom{d+\ell_r}{k} \sum_{c=0}^{k-\ell_r} \binom{k}{c} (|\Sigma|-2)^c \left(\sqrt{2}\left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)\right)^m. \quad (6.16)
\end{aligned}$$

Recall that $m \leq \frac{d-\ell_r-c}{2}$. Also note that $\sqrt{2}\left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right) = \left(\frac{|\Sigma|-2}{\gamma-1}\right)^2$. So, by Inequality 6.16, we have

$$\begin{aligned}
T(d,d) &\leq \sum_{k=\ell_r}^{d} 2^{\frac{2d-k+\ell_r}{2}} \binom{d+\ell_r}{k} \left(\frac{|\Sigma|-2}{\gamma-1}\right)^{d-\ell_r} \sum_{c=0}^{k-\ell_r} \binom{k}{c} (\gamma-1)^c \\
&\leq 2^{d+\frac{\ell_r}{2}} \left(\frac{|\Sigma|-2}{\gamma-1}\right)^{d-\ell_r} \sum_{k=\ell_r}^{d} \binom{d+\ell_r}{k} \left(\frac{\gamma}{\sqrt{2}}\right)^k \\
&\leq 2^{d+\frac{\ell_r}{2}} \left(\frac{|\Sigma|-2}{\gamma-1}\right)^{d-\ell_r} \left(1+\frac{\gamma}{\sqrt{2}}\right)^{d+\ell_r}. \\
&= \left(\sqrt[4]{8}\left(1+\sqrt{|\Sigma|-1}\right)\left(\sqrt{2}+\gamma\right)\right)^d \left(\frac{\sqrt{2}+\gamma}{\sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)}\right)^{\ell_r} \quad (6.17)
\end{aligned}$$

If $|\Sigma| = 2$, then by Inequality 6.17 and the fact that $\ell_r \leq d$, $T(d,d) \leq \left(4+3\sqrt{2}\right)^d$. Otherwise, $\sqrt{2}+\gamma \leq \sqrt[4]{2}\left(1+\sqrt{|\Sigma|-1}\right)$ and hence $T(d,d) \leq \left(\sqrt[4]{8}\left(1+\sqrt{|\Sigma|-1}\right)\left(\sqrt{2}+\gamma\right)\right)^d$ by Inequality 6.17. This completes the proof. □

Proof of Lemma 5.2: 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:

$$T'(d,b) \leq \sum_{w} \sum_{k=\ell(w)}^{b} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \, T'(d, \min\{b-k, k-\ell(w)-c\}), \quad (6.18)$$

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 6.18 and the induction hypothesis,

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

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

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

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

This finishes the proof. □

Proof of Theorem 5.3: 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 6.18, we have

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

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

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

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

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

$$\sum_{k=\ell(w)}^{d} \binom{d+\ell(w)}{k} \sum_{c=0}^{k-\ell(w)} \binom{k}{c} (|\Sigma|-2)^c \binom{d+b'}{b'} (|\Sigma|-1)^{b'} \, 2^{2\nu b'} \leq \left(\frac{\sqrt{27} + \sqrt{27(|\Sigma|-1)}}{2} \cdot (\mu+1) \right)^d.$$

This finishes the proof. □