A Three-String Approach to the Closest String Problem

Zhi-Zhong Chen* Bin Ma† Lusheng Wang‡

Abstract

Given a set of n strings of length L and a radius d, the closest string problem (CSP for short) asks for a string t_{sol} that is within a Hamming distance of d to each of the given strings. It is known that the problem is NP-hard and its optimization version admits a polynomial time approximation scheme (PTAS). Parameterized algorithms have been then developed to solve the problem when d is small. In this paper, with a new approach (called the *3-string approach*), we first design a parameterized algorithm for binary strings that runs in $O\left(nL + nd^3 6.731^d\right)$ time, while the previous best runs in $O\left(nL + nd8^d\right)$ time. We then extend the algorithm to arbitrary alphabet sizes, obtaining an algorithm that runs in time $O\left(nL + nd1.612^d\left(|\Sigma| + \beta^2 + \beta - 2\right)^d\right)$, where $|\Sigma|$ is the alphabet size and $\beta = \alpha^2 + 1 - 2\alpha^{-1} + \alpha^{-2}$ with $\alpha = \sqrt[3]{\sqrt{|\Sigma| - 1} + 1}$. This new time bound is better than the previous best for small alphabets, including the very important case where $|\Sigma| = 4$ (i.e., the case of DNA strings).

Keywords: Computational biology, the closest string problem, fixed-parameter algorithms.

1 Introduction

An instance of the closest string problem (CSP for short) is a pair $(\mathcal{S}, d)$, where $\mathcal{S}$ is a set of strings of the same length L and d is a nonnegative integer (called the *radius*). The objective is to find a string t_{sol} of length L such that $d(t_{sol}, s) \leq d$ for every $s \in \mathcal{S}$. We call t_{sol} a *center string* of radius d for the strings in $\mathcal{S}$. In the optimization version of the problem, only $\mathcal{S}$ is given and the objective is to find the minimum d such that a center string of radius d exists for the strings in $\mathcal{S}$.

The problem finds a variety of applications in bioinformatics, such as universal PCR primer design [16, 14, 5, 22, 11, 26], genetic probe design [14], antisense drug design [14, 4], finding unbiased

*Corresponding author. Department of Mathematical Sciences, Tokyo Denki University, Ishizaka, Hatoyama, Hiki, Saitama, 359-0394, Japan. Email: zzchen@mail.dendai.ac.jp. Phone: +81-49-296-5249. Fax: +81-49-296-7072.

†School of Computer Science, University of Waterloo, 200 University Ave. W, Waterloo, ON, Canada N2L3G1. Email: binma@uwaterloo.ca. Phone: +1-519-888-4567. Fax: +1-519-885-1208.

‡Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR. Email: lwang@cs.cityu.edu.hk. Phone: +852-2788-9820. Fax: +852-2788-8614.

consensus of a protein family [1], and motif finding [14, 11, 24, 3, 8]. Consequently, the problem has been extensively studied in computational biology [14, 15, 18, 12, 20, 11, 19, 13, 7, 10, 23, 4, 21, 24].

The problem is known to be NP-complete [9, 14]. Early attempts to solve this problem mainly focused on approximation algorithms. These include the first non-trivial approximation algorithm with ratio 4/3 [14] and a polynomial time approximation scheme (PTAS) [15]. The time complexity of the PTAS was further improved in [18] and [17]. The main concern of using the PTAS algorithms is that their time complexity is too high. Even with the latest improvement made in [17], the time complexity for achieving an approximation ratio of $1+\epsilon$ is $O(Ln^{O(\epsilon^{-2})})$.

Another approach to the solving of CSP is via parameterized algorithms. A parameterized algorithm computes an exact solution of a problem with time complexity $f(k) \cdot n^c$, where c is a constant, n is the problem size, k is a parameter naturally associated to the input instance, and f is any function [6]. The argument is that if k is typically small for natural instances of the problem, the problem may still be solvable in acceptable time complexity despite that f may be a super-polynomial function.

For the special case of CSP where $d = 1$, Stojanovic *et. al* [23] designed a linear time algorithm. Gramm *et. al* [12] proposed the first parameterized algorithm with time complexity $O\left(nL + n(d+1)^{d+1}\right)$. Ma and Sun [17] gave an algorithm with running time $O\left(nL + nd \cdot 16^d(|\Sigma|-1)^d\right)$, which is the first polynomial time algorithm when d is logarithmic in the input size and the alphabet size $|\Sigma|$ is a constant. Wang and Zhu [25] improved the time complexity to $O(nL + nd2^{3.25d}(|\Sigma|-1)^d)$. Chen and Wang [2] further improved the time complexity to $O\left(nL + nd8^d\right)$ for binary strings and to $O\left(nL + nd\left(\sqrt{2}|\Sigma| + \sqrt[4]{8}\left(\sqrt{2}+1\right)\left(1+\sqrt{|\Sigma|-1}\right) - 2\sqrt{2}\right)^d\right)$ for non-binary strings. Independently, Zhao and Zhang [27] provided an algorithm with running time $O\left(nL + nd\left(2|\Sigma| + 4\sqrt{|\Sigma|-1}\right)^d\right)$. Note that the algorithm in [2] outperforms the algorithm in [27] for any kind of strings.

In this paper, we introduce a new approach (called the *3-string approach*) and use it to design new parameterized algorithms for the problem. Roughly speaking, with this approach, our algorithm starts by carefully selecting three of the input strings and using them to guess a portion of the output center string. In contrast, all previous algorithms were based on the *2-string approach*, with which the algorithms start by carefully selecting two of the input strings and using them to guess a portion of the output center string. Intuitively speaking, the 3-string approach is better, because it enables the algorithm to guess a larger portion of the output center string in the beginning. The new parameterized algorithm for binary strings runs in $O\left(nL + nd^3 6.731^d\right)$ time, while the previous best runs in $O\left(nL + nd8^d\right)$ time. We want to emphasize that $O\left(nL + nd8^d\right)$ seems to be the best possible time complexity achievable by algorithms based on the 2-string approach. We then extend the algorithm to arbitrary strings, obtaining an algorithm that runs in time

$O\left(nL + nd1.612^d \left(|\Sigma| + \beta^2 + \beta - 2\right)^d\right)$, where $\beta = \alpha^2 + 1 - 2\alpha^{-1} + \alpha^{-2}$ with $\alpha = \sqrt[3]{\sqrt{|\Sigma| - 1} + 1}$. In particular, in the very important case where $|\Sigma| = 4$ (i.e., the case of DNA strings), our algorithm runs in $O\left(nL + nd13.183^d\right)$ time, while the previous best runs in $O\left(nL + nd13.921^d\right)$ time.

The remainder of this paper is organized as follows. Section 2 defines a few notations frequently used in the paper. Section 3 reviews the algorithm in [2], which will be helpful for the presentation of the new algorithm. Section 4 details our algorithm for binary strings. Section 5 then extends the algorithm to general alphabets.

2 Notations

Throughout this paper, Σ denotes a fixed alphabet and a string always means one over Σ. For each positive integer k, $[1..k]$ denotes the set $\{1, 2, \ldots, k\}$. For a string s, $|s|$ denotes the length of s. For each $i \in [1..|s|]$, $s[i]$ denotes the letter of s at its i-th position. Thus, $s = s[1]s[2]\ldots s[|s|]$. A position set of a string s is a subset of $[1..|s|]$. For two strings s and t of the same length, $d(s, t)$ denotes their Hamming distance. For a binary string s, $\bar{s}$ denotes the complement string of s, where $\bar{s}[i] \neq s[i]$ for every $i \in [1..|s|]$.

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 $h \geq 2$, $k \geq 1$, 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 $\{s_1 \equiv s_2 \equiv \cdots \equiv s_h\} \cap \{s_h \not\equiv t_1 \not\equiv t_2 \not\equiv \cdots \not\equiv t_k\} \cap \{t_k \equiv u_1 \equiv u_2 \equiv \cdots \equiv u_\ell\}$.

Another useful concept is that of a *partial string*, which is a string whose letters are only known at its certain positions. If s is a string of length L and P is a position set of s, then $s|_P$ denotes the partial string of length L such that $s|_P[i] = s[i]$ for each position $i \in P$ but $s|_P[j]$ is unknown for each position $j \in [1..L] \setminus P$. Let t be another string of length L. For a subset P of $[1..L]$, the *distance* between $s|_P$ and $t|_P$ is $|\{i \in P \mid s[i] \neq t[i]\}|$ and is denoted by $d(s|_P, t|_P)$. For two disjoint position sets P and Q of s, $s|_P + t|_Q$ denotes the partial string $r|_{P \cup Q}$ such that

$$r|_{P\cup Q}[i] = \begin{cases} s[i], & \text{if } i \in P; \\ t[i], & \text{if } i \in Q. \end{cases}$$

At last, when an algorithm exhaustively tries all possibilities to find the right choice, we say that the algorithm *guesses* the right choice.

3 Previous Algorithms and a New Lemma

In this section we familiarize the readers with the basic ideas in the previously known parameterized algorithms for CSP, as well as introduce two technical lemmas that are needed in this paper. All previously known algorithms use the bounded search tree approach for parameterized algorithm design. We explain the ideas based on the algorithm given in [2]. We call the approach used in previous algorithms the *2-string approach* in contrast to the 3-string approach introduced in this paper.

Let $(\mathcal{S}, d)$ be an instance of CSP. Let s_1, s_2, ..., s_n be the strings in $\mathcal{S}$, L be the length of each string in $\mathcal{S}$, and t_{sol} be any solution to $(\mathcal{S}, d)$. The idea is to start with a candidate string t with $d(t, t_{sol}) \leq d$. Using some strategies, the algorithm guesses the letters of t_{sol} at some positions (by trying all legible choices), and modify the letters of t to those of t_{sol} at the positions. This procedure is applied iteratively to eventually change t to t_{sol}. The "guessing" causes the necessity of using a search tree, whose size is related to (1) the number of choices to guess from in each iteration (the degree of each tree node) and (2) the total number of iterations (the height of the tree).

At the beginning of each iteration, the algorithm knows the set $\mathcal{S}$ of input strings, the radius d, the current candidate string t, and an upper bound b on the remaining distance $d(t, t_{sol})$. In addition, if a position in $[1..L]$ has been considered for modification in a previous iteration, it is not helpful to modify it again in the current iteration. Thus, we further record a position set P, at which no further modification is allowed. This gives an *extended closest string problem* (ECSP for short) formerly defined in [2]. An instance of ECSP is a quintuple $\langle \mathcal{S}, d; t, P, b \rangle$, as defined above. A solution of the instance is a string t_{sol} of length L satisfying the following conditions:

1. $t_{sol}|_P = t|_P$.
2. $d(t_{sol}, t) \leq b$.
3. For every string $s \in \mathcal{S}$, $d(t_{sol}, s) \leq d$.

Obviously, to solve CSP for a given instance $\langle \mathcal{S}, d \rangle$, it suffices to solve ECSP for the instance $\langle \mathcal{S} \setminus \{t\}, d; t, \emptyset, d \rangle$, where t is an arbitrary string in $\mathcal{S}$ and $\emptyset$ is the empty set.

The difference between the algorithms in [12], [17] and [2] exists in the guessing strategy in each iteration. In each iteration, if t is not a solution yet, then there must be a string s such that $|\{s \not\equiv t\}| > d$. Since $d(t, t_{sol}) \leq d$, for each subset R of $\{s \not\equiv t\}$ with $|R| = d + 1$, there must be at least one position $i \in R$ with $t_{sol}[i] = s[i]$. The algorithm in [12] first chooses an arbitrary subset R of $\{s \not\equiv t\}$ with $|R| = d + 1$, then simply guesses one position $i \in R$, and further changes $t[i]$ to $s[i]$. This reduces b by one. Thus, the degree of the search tree is $d + 1$ and the height of the tree is d. The search tree size is hence bounded by $(d + 1)^{d+1}/d$.

The algorithm in [17] guesses the partial string $t_{sol}|_{\{s \not\equiv t\}}$ (by carefully enumerating all legible choices) and changes t to $t|_{\{s \equiv t\}} + t_{sol}|_{\{s \not\equiv t\}}$. This gives a much greater degree of the search tree than the algorithm in [12]. However, it was shown in [17] that this strategy at least halves the parameter b for the next iteration. Thus, the height of the tree is at most $O(\log d)$. The search tree size can then be bounded by $(O(|\Sigma|))^d$, which is polynomial when $d = O(\log(nL))$ and $|\Sigma|$ is a constant.

The guessing strategy was further refined in [2] as follows. Recall that P is the set of positions of t that have been modified in previous iterations. Suppose there are k positions in $\{s \not\equiv t\} \setminus P$ where t_{sol} and t differ. Out of these k positions, suppose there are $c \leq k$ positions where t_{sol} is different from both t and s. Then, at each of the c positions, we need to guess the letter of t_{sol} from only $|\Sigma| - 2$ choices. Moreover, at the other $k - c$ positions, we do not need to guess and can simply let t_{sol} be equal to s. So, when c is small, the degree of the search tree node is reduced. On the other hand, when c is large, the following lemma proved in [2] shows that b is greatly reduced for the next iteration, yielding a smaller search tree height. With this lemma, a further improved time complexity is proved in [2]. The algorithm is given in Figure 1.

Lemma 3.1 *[2] Let $\langle \mathcal{S}, d; t, P, b\rangle$ be an instance of ECSP with a solution t_{sol}. Suppose that s is a string in $\mathcal{S}$ with $d(t, s) = d + \ell > d$. Let k be the number of positions in $\{s \not\equiv t\} \setminus P$ where t_{sol} is different from t. Let c be the number of positions in $\{s \not\equiv t\} \setminus P$ where t_{sol} is different from both t and s. Let b' be the number of positions in $[1..L] \setminus (P \cup \{s \not\equiv t\})$ where t_{sol} is different from t. Then, $b' \leq b - k$ and $b' \leq k - \ell - c$. Consequently, $b' \leq \frac{b-\ell-c}{2}$.*

The execution of the 2-string algorithm on input $\langle \mathcal{S}, d; t, P, b\rangle$ can be modeled by a tree $\mathcal{T}$ in which the root corresponds to $\langle \mathcal{S}, d; t, P, b\rangle$, 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 $\langle \mathcal{S}, d; t, P, b\rangle$. By the construction of the algorithm, each non-leaf 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 *depth* of a node u in $\mathcal{T}$ is the distance between the root and u in $\mathcal{T}$. In particular, the depth of the root is 0. The *depth* of $\mathcal{T}$ is the maximum depth of a node in $\mathcal{T}$.

During the execution of the algorithm on input $\langle \mathcal{S}, d; t, P, b\rangle$, d does not change but the other parameters may change. We use $\mathcal{S}_u$, t_u, P_u, and b_u to denote the values of $\mathcal{S}$, t, P, and b when the algorithm enters the node u (i.e., makes the recursive call corresponding to u). Moreover, we use s_u and ℓ_u to denote the string s and the integer ℓ computed in Steps 2 and 3 of the algorithm at node u.

Let $T(d, b_u)$ denote the size of the subtree rooted at u. The following lemma was proved in [2]:

The 2-String Algorithm

Input: An instance $\langle \mathcal{S}, d; t, P, b \rangle$ of ECSP.

Output: A solution to $\langle \mathcal{S}, d; t, P, b \rangle$ 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. If $d = b$, then find a string $s \in \mathcal{S}$ such that $d(t, s)$ is maximized over all strings in $\mathcal{S}$; otherwise, find an arbitrary string $s \in \mathcal{S}$ such that $d(t, s) > d$.

3. Let $\ell = d(t, s) - d$ and $R = \{s \not\equiv t\} \setminus P$.

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

5. *Guess* $t_{sol}|_R$ by the following steps:

5.1 *Guess* two sets X and Y such that $Y \subseteq X \subseteq R$, $\ell \leq |X| \leq b$, and $|Y| \leq |X| - \ell$. (*Comment:* $|X|$ and $|Y|$ correspond to k and c in Lemma 3.1, respectively.)

5.2 For each $i \in Y$, *guess* a letter z_i different from both $s[i]$ and $t[i]$. Let the partial string $\hat{s}|_Y$ be such that $\hat{s}|_Y[i] = z_i$ for all $i \in Y$.

5.3 Let $t_{sol}|_R = \hat{s}|_Y + s|_{X \setminus Y} + t|_{R \setminus X}$.

6. Let $t' = t_{sol}|_R + t|_{[1..|t|] \setminus R}$ and $b' = \min\{b - |X|, |X| - \ell - |Y|\}$.

7. Solve $\langle \mathcal{S} - \{s\}, d; t', P \cup R, b' \rangle$ recursively.

8. Return NULL.

Figure 1: The algorithm given in [2].

Lemma 3.2 *[2] For each descendant u of r in $\mathcal{T}$,*

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

We next prove a new lemma for the 2-string algorithm:

Lemma 3.3 *For each node u at depth $h \geq 2$ in $\mathcal{T}$,*

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

Proof. If the depth of $\mathcal{T}$ is at most 1, then the lemma is trivially true. So, suppose that the depth of $\mathcal{T}$ is at least 2. Consider an arbitrary node u whose depth in $\mathcal{T}$ is at least 2. Let u_1, u_2, $\ldots$, u_{h-1} be the nodes (other than r and u) we meet on the way from r to u in $\mathcal{T}$. For convenience, let $u_0 = r$ and $u_h = u$. For each integer i with $0 \leq i \leq h - 1$, let $k_i = d(t_{u_i}, t_{u_{i+1}})$. Then, by the computation of b' in Step 6 of the 2-string algorithm, $k_0 \geq \ell_r + b_{u_1}$ and $b_{u_i} \geq k_i + b_{u_{i+1}}$ for each $1 \leq i \leq h - 1$. So, $k_0 \geq \ell_r + \sum_{i=1}^{h-1} k_i + b_u$. Again, by the computation of b' in Step 6 of the 2-string

algorithm, $b_{u_i} \geq 2b_{u_{i+1}}$ and $k_i \geq b_{u_{i+1}}$ for each $0 \leq i \leq h-1$. So, for each $0 \leq i \leq h-1$, $b_{u_i} \geq 2^{h-i}b_u$ and $k_i \geq 2^{h-i-1}b_u$. Now, $d(t_r, t_u) = \sum_{i=0}^{h-1} k_i \geq \ell_r + b_u + 2\sum_{i=1}^{h-1} k_i \geq \ell_r + b_u + 2b_u \sum_{i=1}^{h-1} 2^{h-i-1} = \ell_r + (2^h - 1)b_u$. Thus, by Lemma 3.2, $T(d, b_u) \leq \binom{d-(2^{h-1}-1)b_u}{b_u} \cdot \left(|\Sigma| + 2\sqrt{|\Sigma|-1}\right)^{b_u}$. □

4 The 3-String Algorithm for the Binary Case

In addition to Lemma 3.3, the improvements made in this paper mainly come from a new strategy to collapse the first two levels (the root and its children) of the search tree into a single level. We call this new approach the *3-string approach*. In this section, we demonstrate the approach by designing an algorithm for the binary-alphabet case because of its simplicity. In Section 5, we will extend the algorithm to the general case.

Note that given an instance $(\mathcal{S}, d)$ of CSP such that there are at most two strings in $\mathcal{S}$, we can solve it trivially in linear time. So, we hereafter assume that each given instance $(\mathcal{S}, d)$ of the problem satisfies that $|\mathcal{S}| \geq 3$.

4.1 First Trick: *Guessing* Ahead

Let us briefly go through the first two levels of the search tree $\mathcal{T}$ of the 2-string algorithm. The algorithm starts by initializing t_r to be an arbitrary string in $\mathcal{S}$. At the root r, it finds a string $s_r \in \mathcal{S}$ that maximizes $d(t_r, s_r)$. It then uses s_r to modify t_r and further enters a child node u of r (i.e., makes a recursive call). Note that t_r has become t_u at u. Suppose that the subtree $\mathcal{T}_u$ of $\mathcal{T}$ rooted at u contains a solution t_{sol}. The algorithm then finds a string $s_u \in \mathcal{S}$ such that $d(t_u, s_u) > d$ in Step 2. Note that $P_r = \{t_r \not\equiv s_r\}$ and $P_u \setminus P_r = \{t_r \equiv s_r \not\equiv s_u\}$.

The main idea of the 3-string approach is that we *guess* s_u at the very beginning of the algorithm, instead of finding it in the second-level recursion. This will immediately increase the time complexity by a factor of $n-2$ because there are $n-2$ choices of s_u. However, with all of the three strings t_r, s_r, and s_u in hand, we will be able to *guess* $t_{sol}|_{P_u}$ easier, which leads to a better time complexity. The trade-off is a good one when d is large. In fact, we do not even need to trade off. In Subsection 4.2, we will introduce another trick to get rid of this factor of $n-2$.

Lemma 4.1 *Let $(\mathcal{S}, d)$ be an instance of the binary case of CSP. Suppose that t_r, s_r, and s_u are three strings in $\mathcal{S}$ and t_{sol} is a solution of $(\mathcal{S}, d)$. Let $P_r = \{t_r \not\equiv s_r\}$, $R_u = \{t_r \equiv s_r \not\equiv s_u\}$, $P' = \{t_{sol} \not\equiv s_u\} \cap P_r$, $R' = \{t_{sol} \not\equiv t_r\} \cap R_u$, and $B = \{t_r \equiv s_r \equiv s_u \not\equiv t_{sol}\}$. Then, $|P_r| + |P'| + |R_u| + |R'| \leq 3d - 3|B|$.*

Proof. Because t_{sol} is a solution we have

$$d(t_{sol}, t_r) + d(t_{sol}, s_r) + d(t_{sol}, s_u) \leq 3d. \tag{4.1}$$

Because t_r and s_r differ in P_r, each position in P_r contributes at least 1 to the left-hand side of Eq. 4.1. Meanwhile, each position in $P' \subseteq P_r$ contributes 2 (cf. Figure 2). Thus, the total contribution in P_r is $|P_r| + |P'|$. Similarly, each position in R_u contributes at least 1 and each position in $R' \subseteq R_u$ contributes 2. So, the total contribution in R_u is $|R_u| + |R'|$. Finally, each position in B contributes 3. From Eq. 4.1, we have $(|P_r| + |P'|) + (|R_u| + |R'|) + 3|B| \leq 3d$. The lemma is proved. □

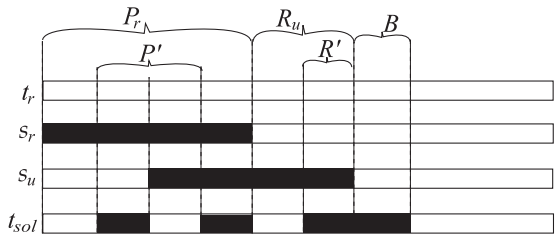

Figure 2: Strings t_r, s_r, s_u, and t_{sol} in Lemma 4.1, where for each position $i \in [1..|t_r|]$, two of the strings have the same letter at the ith position if and only if the two strings are illustrated in the same color at the ith position.

Lemma 4.1 suggests that we can construct $t_{sol}|_{P_r \cup R_u}$ by guessing P' and R', then use $\bar{s}_u|_{P'} + s_u|_{P_r \setminus P'} + \bar{t}_r|_{R'} + t_r|_{R_u \setminus R'}$. For positions in B, we can call the 2-string algorithm to solve it (recursively). Because of the bound $|P_r| + |P'| + |R_u| + |R'| \leq 3d - 3|B|$, we either have a smaller number of choices for guessing P' and R', or have a smaller $|B|$ which makes the recursive call of the 2-string algorithm easier. Likely this will lead to a more efficient algorithm than doing the 2-string algorithm from the beginning. We detail the 3-string algorithm for the binary case of CSP in Figure 3.

To analyze the time complexity of the algorithm, we need a simple technical lemma:

Lemma 4.2

$$\binom{p}{k} \leq \left(\frac{1+\sqrt{5}}{2}\right)^{k+p} \approx 1.618^{k+p} \tag{4.2}$$

Proof. The lemma is clearly true when $k = 0$ or p. Consider an arbitrary integer k with $1 \leq k \leq p-1$. Let $\alpha = \frac{k}{p}$. By Stirling's formula, $\binom{p}{k} \leq \alpha^{-\alpha p}(1-\alpha)^{-(1-\alpha)p}$. So, to finish the proof, it suffices to show that $\alpha^{-\alpha}(1-\alpha)^{-(1-\alpha)} \leq \left(\frac{1+\sqrt{5}}{2}\right)^{1+\alpha}$. Consider the function $f(\alpha) = (1+\alpha)\log\frac{1+\sqrt{5}}{2} + \alpha\log\alpha + (1-\alpha)\log(1-\alpha)$. It remains to prove that $f(\alpha) \geq 0$. By calculus, we can see that (1) $f'(\alpha) = 0$ when $\alpha = \frac{2}{3+\sqrt{5}}$, (2) $f'(\alpha) < 0$ when $\alpha < \frac{2}{3+\sqrt{5}}$, and (3) $f'(\alpha) > 0$ when $\alpha > \frac{2}{3+\sqrt{5}}$. So, $f(\alpha)$ achieves the minimum value at $\alpha = \frac{2}{3+\sqrt{5}}$. Thus, we only need to prove that $f(\frac{2}{3+\sqrt{5}}) \geq 0$. Since the last inequality can be easily verified, the lemma is proved. □

The next lemma slightly strengthens Lemma 4.2 when $p \geq 3k$.

The 3-String Algorithm for the Binary Case

Input: An instance $(\mathcal{S}, d)$ of the binary case of CSP.

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

1. Select an arbitrary string $t_r \in \mathcal{S}$.

2. If there is no $s \in \mathcal{S}$ with $d(t_r, s) > d$, then output t_r and halt.

3. Find a string $s_r \in \mathcal{S}$ such that $d(t_r, s_r)$ is maximized. (*Comment:* If a solution t_{sol} exists, then $d(t_r, s_r) \leq d(t_r, t_{sol}) + d(t_{sol}, s_r) \leq d + d = 2d$.)

4. Let $P_r = \{t_r \not\equiv s_r\}$. If $|P_r| > 2d$, then output NULL and halt.

5. *Guess* a string $s_u \in \mathcal{S} \setminus \{t_r, s_r\}$.

6. *Guess* a subset P' of P_r with $|P'| \leq d$.

7. Let $t' = \bar{s}_u|_{P'} + s_u|_{P_r \setminus P'} + t_r|_{[1..|t_r|] \setminus P_r}$.

8. If there is no $s \in \mathcal{S}$ with $d(t', s) > d$, then output t' and halt.

9. If $d(t', t_r) \leq d$, $d(t', s_r) \leq d$, and $d(t', s_u) > d$, then perform the following steps:

9.1. *Guess* a subset R' of R_u such that $|R'| \leq 3d - |P_r| - |R_u| - |P'|$, where $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. (*Comment:* The upper bound on $|R'|$ used in this step comes from Lemma 4.1.)

9.2. Let $t = t'|_{P_r} + \bar{t}_r|_{R'} + t_r|_{[1..|t_r|] \setminus (P_r \cup R')}$.

9.3. If $d(t, t_r) \leq d$, $d(t, s_r) \leq d$, and $d(t, s_u) \leq d$, then perform the following steps:

9.3.1. Compute $\ell_r = d(t_r, s_r) - d$, $k_1 = d(t_r, t')$, $b_1 = \min\{d - k_1, k_1 - \ell_r\}$, $k_2 = |R'|$, $\ell_2 = d(t', s_u) - d$, and $b_2 = \min\{b_1 - k_2, k_2 - \ell_2, (3d - |P_r| - |R_u| - |P'| - |R'|)/3\}$. (*Comment:* Obviously, $b_1 = \min\{d - k_1, k_1 - \ell_r\}$ mimics the computation of b' in Step 6 of the 2-string algorithm on input $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$, while $b_2 \leq \min\{b_1 - k_2, k_2 - \ell_2\}$ mimics the computation of b' in Step 6 of the 2-string algorithm on input $\langle \mathcal{S} \setminus \{t_r, s_r\}, d; t', P_r, b_1\rangle$. Moreover, $b_2 \leq (3d - |P_r| - |R_u| - |P'| - |R'|)/3$ follows from Lemma 4.1.)

9.3.2. Call the 2-string algorithm to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t, P_r \cup R_u, b_2\rangle$.

10. Output NULL and halt.

Figure 3: The 3-string algorithm for the binary case.

Lemma 4.3 *Suppose that $p \geq 3k$. Then,*

$$\binom{p}{k} \leq \left(\sqrt[4]{6.75}\right)^{k+p} \approx 1.612^{k+p} \tag{4.3}$$

PROOF. The proof is similar to that of Lemma 4.2. What we need to prove here is that $\alpha^{-\alpha}(1-\alpha)^{-(1-\alpha)} \leq \left(\sqrt[4]{6.75}\right)^{1+\alpha}$. Consider the function $f(\alpha) = (1+\alpha)\log\sqrt[4]{6.75}+\alpha\log\alpha+(1-\alpha)\log(1-\alpha)$. Our goal is to prove that $f(\alpha) \geq 0$. By calculus, we can see that (1) $f'(\alpha) = 0$ when $\alpha = \frac{1}{1+\sqrt[4]{6.75}}$, (2) $f'(\alpha) < 0$ when $\alpha < \frac{1}{1+\sqrt[4]{6.75}}$, and (3) $f'(\alpha) > 0$ when $\alpha > \frac{1}{1+\sqrt[4]{6.75}}$. Note that $\frac{1}{1+\sqrt[4]{6.75}} > \frac{1}{3} \geq \alpha$ for $p \geq 3k$. So, when $p \geq 3k$, $f(\alpha)$ achieves the minimum value at $\alpha = \frac{1}{3}$. Thus, it remains to prove that $f(\frac{1}{3}) \geq 0$. Since the last inequality can be easily verified, the lemma is proved. □

Theorem 4.4 *The binary case of CSP can be solved in $O\left(nL + n^2d^26.731^d\right)$ time.*

PROOF. For convenience, we use $\mathcal{A}_2$ (respectively, $\mathcal{A}_3$) to denote the 2-string (respectively, 3-string) algorithm. If $\mathcal{A}_3$ halts in Step 2, $\mathcal{A}_3$ is clearly correct. So, we further assume that $\mathcal{A}_3$ does not halt in Step 2. Then, we may assume that the string s_r found by $\mathcal{A}_3$ in Step 3 is the same as the string s found by $\mathcal{A}_2$ in Step 2 on input $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$. If $d \leq 25$, then clearly $\mathcal{A}_3$ runs in $O(n^2)$ time and the theorem is true. So, we further assume that $d \geq 26$.

Let $\mathcal{T}_2$ be the search tree of $\mathcal{A}_2$ on input $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$. Note that each grandchild v of the root r in $\mathcal{T}_2$ corresponds to an instance $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t_v, P_v, b_v\rangle$ of ECSP, where u is the parent of v in $\mathcal{T}_2$. By the construction of $\mathcal{A}_3$, it is clear that for each such v, $\mathcal{A}_3$ can first guess s_u (in Step 5), then correctly compute t_v (as t in Step 9.2), P_v (trivially as $P_r \cup R_u$), and b_v (as b_2 in Step 9.3.1), and finally call $\mathcal{A}_2$ to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t_v, P_v, b_v\rangle$ in Step 9.3.2. Thus, if r has a grandchild v in $\mathcal{T}_2$ such that a solution of the instance $\langle \mathcal{S} - \setminus\{t_r\}, d; t_r, \emptyset, d\rangle$ is found at a descendant of v in $\mathcal{T}_2$, then $\mathcal{A}_3$ will find a solution, too.

It remains to consider the case where a solution of the instance $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$ is found at a child v of r in $\mathcal{T}_2$. Note that v is a leaf of $\mathcal{T}_2$ and the solution found at v is t_v. For this v, $\mathcal{A}_3$ will first guess an *arbitrary* string s_u (in Step 5), then correctly compute t_v (as t' in Step 7), and finally output t_v in Step 8. So, $\mathcal{A}_3$ is correct in this case, too.

Next we estimate the time complexity of $\mathcal{A}_3$. The execution of $\mathcal{A}_3$ on input $(\mathcal{S}, d)$ can be modeled by a *search tree* $\mathcal{T}$ as follows. The root r of $\mathcal{T}$ corresponds to $(\mathcal{S}, d)$. For each pair (s_u, P') of possible outcomes of the guesses made in Steps 5 and 6 such that $\mathcal{A}_3$ does not execute Step 9.1, r has a child corresponding to (s_u, P'). Similarly, for each triple (s_u, P', R') of possible outcomes of the guesses made in Steps 5, 6, and 9.1 such that $\mathcal{A}_3$ executes Step 9.1, r has a child corresponding to (s_u, P', R'). Moreover, for every child v corresponding to a triple (s_u, P', R') such that $\mathcal{A}_3$ executes Step 9.3.2, v has descendants in $\mathcal{T}$ so that the subtree of $\mathcal{T}$ rooted at v is the same as the search tree of $\mathcal{A}_2$ on input $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t, P_r \cup R_u, b_2\rangle$.

Note that each non-leaf node of $\mathcal{T}$ has at least two children in $\mathcal{T}$. So, 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 each string s_u guessed in Step 5, $\mathcal{A}_3$ guesses $P' \subset P_r$ in Step 6 and possibly $R' \subset R_u$ in Step 9.1. For convenience, let $h = 3d - |P_r| - |R_u|$. Then, the number N of children of r in $\mathcal{T}$ that are not leaves of $\mathcal{T}$ satisfies the following inequalities:

$$N \leq \sum_{s_u} \sum_{i=0}^{h} \binom{|P_r|}{i} \sum_{j=0}^{h-i} \binom{|R_u|}{j} \leq \sum_{s_u} \sum_{x=0}^{h} \binom{|P_r| + |R_u|}{x}, \tag{4.4}$$

where i, j, and x correspond to $|P'|$, $|R'|$, and $|P'| + |R'|$ in the algorithm, respectively. So, by Lemmas 3.3 and 4.2, the number N' of leaves at depth 2 or more in $\mathcal{T}$ satisfies the following inequalities:

$$N' \leq \sum_{s_u} \sum_{x=0}^{h} \binom{|P_r| + |R_u|}{x} \binom{d - b_2}{b_2} 4^{b_2} \leq \sum_{s_u} \sum_{x=0}^{h} 1.618^{|P_r|+|R_u|+x} \binom{d - b_2}{b_2} 4^{b_2}. \tag{4.5}$$

By Step 9.3.1, $3b_2 \leq 3d - |P_r| - |R_u| - x$. Thus, we have

$$N' \leq \sum_{s_u} \sum_{x=0}^{h} 1.618^{3d-3b_2} \binom{d - b_2}{b_2} 4^{b_2}.$$

Consider the function $\varphi(b_2) = 1.618^{3d-3b_2}\binom{d-b_2}{b_2}4^{b_2}$. By Step 9.3.1, $b_1 \leq 0.5d$ and $b_2 \leq 0.5b_1 \leq 0.25d$. Since $d \geq 26$, one can verify that $\varphi(b_2 + 1) > \varphi(b_2)$ for $b_2 \leq 0.25d$. Thus, $\varphi(b_2)$ is an increasing function of b_2 for $b_2 \leq 0.25d$. By Stirling's formula, we also have $\binom{0.75d}{0.25d} \leq 3^{0.25d}1.5^{0.5d}$. Of course, $h < 2d$ for $|P_r| > d$. Therefore,

$$N' \leq \sum_{s_u} \sum_{x=0}^{h} 1.618^{2.25d} \binom{0.75d}{0.25d} 4^{0.25d} \leq 2nd \cdot 1.618^{2.25d} 3^{0.25d} 1.5^{0.5d} 4^{0.25d} \leq 2nd \cdot 6.731^d. \tag{4.6}$$

We next bound the number of leaves at depth 1 in $\mathcal{T}$. Clearly, the number M_1 of leaves at depth 1 in $\mathcal{T}$ corresponding to a pair satisfies the following inequalities:

$$M_1 \leq \sum_{s_u} \sum_{i=0}^{d} \binom{|P_r|}{i} \leq \sum_{s_u} 2^{|P_r|} \leq (n-2)4^d, \tag{4.7}$$

where the last inequality holds because Step 4 ensures that $|P_r| \leq 2d$.

On the other hand, the number M_2 of leaves at depth 1 in $\mathcal{T}$ corresponding to a triple satisfies the following inequalities:

$$M_2 \leq \sum_{s_u} \sum_{i=0}^{h} \binom{|P_r|}{i} \sum_{j=0}^{h-i} \binom{|R_u|}{j} \leq \sum_{s_u} \sum_{x=0}^{h} \binom{|P_r| + |R_u|}{x} \binom{d - b_2}{b_2} 4^{b_2}, \tag{4.8}$$

where we simply let $b_2 = h - x$ to ensure that $\binom{d-b_2}{b_2}4^{b_2} \geq 1$. Note that the last bound on M_2 in Eq. 4.8 is the same as the first bound on N' in Eq. 4.5. So, by Eq. 4.6, we also have $M_2 \leq 2nd \cdot 6.731^d$.

Therefore, by Eq. 4.6 and 4.7, the total number of leaves in $\mathcal{T}$ is $N' + M_1 + M_2 \leq 6nd \cdot 6.731^d$. Consequently, $\mathcal{A}_3$ runs in $O\left(nL + n^2d^2 6.731^d\right)$ time, because each node of $\mathcal{T}$ takes $O(nd)$ time. □

In the next subsection, we consider the case where $n > d$. The goal is to reduce the running time of the 3-string algorithm by replacing a factor of $O(n)$ with $O(d)$.

4.2 Second Trick: Avoiding Guessing s_u

The crux is to modify Step 5 of the algorithm in Subsection 4.1 as follows:

> **5.** Find a string $s_u \in \mathcal{S} \setminus \{t_r, s_r\}$ such that $|\{t_r \equiv s_r \not\equiv s_u\}|$ is maximized.

The problem caused by this change is that in Step 9 of the algorithm, we cannot guarantee $d(t', s_u) > d$ any more. Thus, if $d(t', s_u) \leq d$, we want to replace s_u by a new string $\tilde{s}_u$ such that $d(t', \tilde{s}_u) > d$. More specifically, Step 9 of the algorithm in Subsection 4.1 is replaced by the following:

> **9.** If $d(t', t_r) \leq d$ and $d(t', s_r) \leq d$, then perform the following steps:
>
> **9.0.** If $d(t', s_u) \leq d$, then select an arbitrary string $\tilde{s}_u \in \mathcal{S} \setminus \{t_r, s_r, s_u\}$ with $d(t', \tilde{s}_u) > d$, and further let s_u refer to the same string as $\tilde{s}_u$ does. (*Comment:* Since $\max\{d(t', t_r), d(t', s_r), d(t', s_u)\} \leq d$ but t' is not a solution, $\tilde{s}_u$ must exist.)
>
> **9.1 − 9.3.** Same as those of the algorithm in Figure 3, respectively.

The key point here is the following lemma:

Lemma 4.5 *Let t_{sol} be a solution of $(\mathcal{S}, d)$. Consider the time point where the refined algorithm just selected $\tilde{s}_u$ in Step 9.0 but has not let s_u refer to the same string as $\tilde{s}_u$ does. Then, $|P'| < d(t'|_{P_r}, \tilde{s}_u|_{P_r})$. Moreover, $|P_r| + |P'| + |\tilde{R}_u| + |R'| \leq 3d - 3|\tilde{B}|$, where $\tilde{R}_u = \{t_r \equiv s_r \not\equiv \tilde{s}_u\}$, $R' = \{t_{sol} \not\equiv t_r\} \cap \tilde{R}_u$, and $\tilde{B} = \{t_r \equiv s_r \equiv \tilde{s}_u \not\equiv t_{sol}\}$.*

Proof. Let $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. By the modified Step 5, $|\tilde{R}_u| \leq |R_u|$. Since $d(t', s_u) = |P'| + |R_u| \leq d$ and $d(t', \tilde{s}_u) = d(t'|_{P_r}, \tilde{s}_u|_{P_r}) + |\tilde{R}_u| > d$, we have $|P'| < d(t'|_{P_r}, \tilde{s}_u|_{P_r})$.

By Lemma 4.1, $|P_r| + d(t'|_{P_r}, \tilde{s}_u|_{P_r}) + |\tilde{R}_u| + |R'| \leq 3d - 3|\tilde{B}|$. Since $|P'| < d(t'|_{P_r}, \tilde{s}_u|_{P_r})$, we have $|P_r| + |P'| + |\tilde{R}_u| + |R'| \leq 3d - 3|\tilde{B}|$. □

Theorem 4.6 *The refined 3-string algorithm solves the binary case of CSP in $O\left(nL + nd^3 6.731^d\right)$ time.*

PROOF. To see the correctness of the refined algorithm, it suffices to consider the case where $\tilde{s}_u$ is selected in Step 9.0. In this case, by Lemma 4.5, the algorithm can correctly guess $R' = \{t_{sol} \not\equiv t_r\} \cap \tilde{R}_u$ in Step 9.1 because $|R'| \leq 3d - |P_r| - |P'| - |\tilde{R}_u|$. The correctness of the computation of the upper bound b_2 on the remaining distance between t and t_{sol} in Step 9.3.1 again follows from Lemma 4.5. From these facts, it is not hard to see that the refined algorithm is correct.

Since the refined algorithm does not guess s_u, its time complexity seems to be better than that of the algorithm in Subsection 4.1 by a factor of $O(n)$. However, we are unable to prove this, because we cannot simply remove the summation on s_u from Eq. 4.4 in Subsection 4.1. Indeed, instead of Eq. 4.4 in Subsection 4.1, we only have the following inequalities for the refined algorithm:

$$N \leq \sum_{i=0}^{h} \binom{|P_r|}{i} \sum_{j=0}^{h-i} \binom{|R_u|}{j} \leq \sum_{i=0}^{h} \sum_{j=0}^{h-i} \binom{|P_r| + |R_u|}{i+j}, \tag{4.9}$$

where i and j correspond to $|P'|$ and $|R'|$ in the refined algorithm, respectively. The reason why we cannot replace the right-hand side of the last inequality in Eq. 4.9 by $\sum_{x=0}^{h} \binom{|P_r|+|R_u|}{x}$ is that R_u may depend on i in the refined algorithm. Now, we can mimic the analysis in the proof of Theorem 4.4 to show that the refined algorithm runs in $O\left(nL + nd^3 6.731^d\right)$ time. □

The previously best time complexity for the binary case is $O\left(nL + nd8^d\right)$ [2, 27]. The new algorithm is better when d is large (say, ≥ 44). When d is small, the algorithm in the next section is better because its running time for binary strings is $O\left(nL + nd6.911^d\right)$.

5 Extension to Arbitrary Alphabets

There are two main ideas behind the algorithm in Section 4. One is to use Lemma 4.1 to obtain a better bound on $|B|$. The other is to obtain $t_{sol}|_{P_r}$ from $s_u|_{P_r}$ by modifying $s_u|_{P'}$ (instead of obtaining $t_{sol}|_{P_r}$ from $t_r|_{P_r}$ by modifying $t_r|_{P'}$ as in the 2-string algorithm). It is easy to show that Lemma 4.1 still holds for arbitrary alphabets. However, the following lemma is stronger than Lemma 4.1:

Lemma 5.1 *Suppose that t_r, s_r, and s_u are three strings of the same length L and t_{sol} is another string of length L with $d(t_{sol}, t_r) \leq d$, $d(t_{sol}, s_r) \leq d$, and $d(t_{sol}, s_u) \leq d$. Let $P_r = \{t_r \not\equiv s_r\}$, $R_u = \{t_r \equiv s_r \not\equiv s_u\}$, $P' = \{t_{sol} \not\equiv s_u\} \cap P_r$, $R' = \{t_{sol} \not\equiv t_r\} \cap R_u$, $B = \{t_r \equiv s_r \equiv s_u \not\equiv t_{sol}\}$, $C_1 = \{s_u \equiv t_r \not\equiv s_r \not\equiv t_{sol}\}$, $C_2 = \{s_u \equiv s_r \not\equiv t_r \not\equiv t_{sol}\}$, $C_3 = \{t_r \not\equiv s_r \not\equiv s_u \not\equiv t_{sol}\}$, $C_4 = \{s_u \equiv t_{sol} \not\equiv t_r \not\equiv s_r\}$, and $C_5 = \{t_r \equiv s_r \not\equiv s_u \not\equiv t_{sol}\}$. Then,*

$$|P_r| + |P'| + |R_u| + |R'| \leq 3d - 3|B| - \sum_{i=1}^{5} |C_i|.$$

PROOF. Let $b = |B|$ and $c_i = |C_i|$ for all $i \in \{1, \ldots, 5\}$ (cf. Figure 4). For convenience, let $a_1 = |\{t_r \equiv s_u \equiv t_{sol} \not\equiv s_r\}|$, $a_2 = |\{t_r \equiv s_u \not\equiv s_r \equiv t_{sol}\}|$, $a_3 = |\{t_r \equiv t_{sol} \not\equiv s_r \equiv s_u\}|$, $a_4 = |\{t_r \equiv t_{sol} \not\equiv s_r \not\equiv s_u\}|$, $a_5 = |\{s_r \equiv t_{sol} \not\equiv t_r \not\equiv s_u\}|$, $a_6 = |\{s_r \equiv s_u \equiv t_{sol} \not\equiv t_r\}|$, $a_7 = |\{t_r \equiv s_r \equiv t_{sol} \not\equiv s_u\}|$, and $a_8 = |\{t_r \equiv s_r \not\equiv s_u \equiv t_{sol}\}|$. Since $d(t_{sol}, t_r) \leq d$, $d(t_{sol}, s_r) \leq d$, and $d(t_{sol}, s_u) \leq d$,

$$\begin{aligned} d(t_r, t_{sol}) &= a_2 + a_5 + a_6 + a_8 + b + \sum_{i=1}^{5} c_i \leq d, \\ d(s_r, t_{sol}) &= a_1 + a_3 + a_4 + a_8 + b + \sum_{i=1}^{5} c_i \leq d, \\ d(s_u, t_{sol}) &= c_1 + c_2 + c_3 + c_5 + b + a_7 + \sum_{i=2}^{5} a_i \leq d. \end{aligned}$$

Summing up the left-hand and the right-hand sides of the above three inequalities respectively, we have

$$3b + a_1 + a_6 + a_7 + 2a_8 - c_4 + 2\sum_{i=2}^{5} a_i + 3\sum_{i=1}^{5} c_i \leq 3d. \tag{5.1}$$

On the other hand, we also have

$$|P_r| + |R_u| + |P'| + |R'| = \sum_{i=1}^{8} a_i + \sum_{i=1}^{5} c_i + \sum_{i=2}^{5} a_i + \sum_{i=1}^{3} c_i + a_8 + c_5. \tag{5.2}$$

By Eq. 5.1 and 5.2, the lemma holds. □

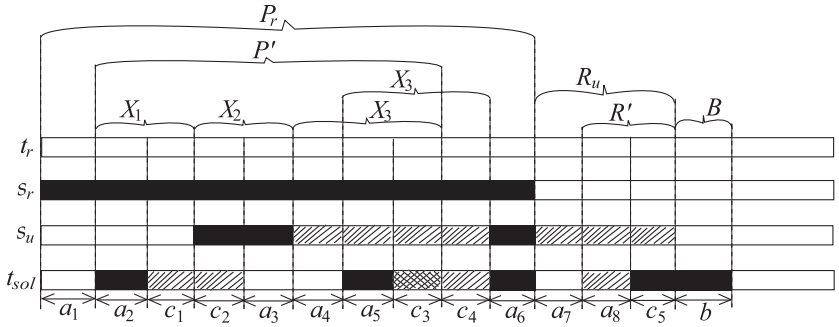

Figure 4: Strings t_r, s_r, s_u, and t_{sol} in Lemma 5.1, where for each position $i \in [1..|t_r|]$, two of the strings have the same letter at the ith position if and only if the two strings are illustrated in the same color or pattern at the ith position.

To understand the remainder of this section, Figure 4 will be very helpful. In addition to the sets defined in Lemma 5.1, we also need the following sets:

- $A_1 = \{t_r \equiv s_u \equiv t_{sol} \not\equiv s_r\}$.

- $A_2 = \{t_r \equiv s_u \not\equiv s_r \equiv t_{sol}\}$.
- $A_3 = \{t_r \equiv t_{sol} \not\equiv s_r \equiv s_u\}$.
- $A_4 = \{t_r \equiv t_{sol} \not\equiv s_r \not\equiv s_u\}$.
- $A_5 = \{s_r \equiv t_{sol} \not\equiv t_r \not\equiv s_u\}$.
- $A_6 = \{s_r \equiv s_u \equiv t_{sol} \not\equiv t_r\}$.
- $A_7 = \{t_r \equiv s_r \equiv t_{sol} \not\equiv s_u\}$.
- $A_8 = \{t_r \equiv s_r \not\equiv s_u \equiv t_{sol}\}$.
- $X_1 = A_2 \cup C_1$.
- $X_2 = C_2 \cup A_3$.
- If $|A_4| \leq |C_4|$, then $X_3 = A_4 \cup A_5 \cup C_3$; otherwise, $X_3 = A_5 \cup C_3 \cup C_4$.

As in the binary case, to compute t_{sol}, our algorithm will first use $t_r|_{P_r \cup R_u}$, $s_r|_{P_r \cup R_u}$, and $s_u|_{P_r \cup R_u}$ to compute $t_{sol}|_{P_r \cup R_u}$, and then call the 2-string algorithm to compute $t_{sol}|_{[1..|t_{sol}|]\setminus(P_r \cup R_u)}$. We here explain how to compute $t_{sol}|_{P_r \cup R_u}$. Note that for each $i \in \{1, \ldots, 8\}$, $|A_i|$ corresponds to a_i in Figure 4. As can be seen from the figure, if we know A_1 through A_8, then we can use the three strings t_r, s_r, and s_u to figure out $t_{sol}|_A$, where $A = \bigcup_{i=1}^{8} A_i$. Unfortunately, we do not know A_1 through A_8. Our idea is then to *guess* X_1, X_2, X_3, and R'. In this way, since we know $A_1 \cup X_1 = \{t_r \equiv s_u \not\equiv s_r\}$, $A_6 \cup X_2 = \{s_r \equiv s_u \not\equiv t_r\}$, and either $C_4 \cup X_3 = \{t_r \not\equiv s_r \not\equiv s_u\}$ or $A_4 \cup X_3 = \{t_r \not\equiv s_r \not\equiv s_u\}$, we can find out A_1, A_6, A_7, and either C_4 or A_4. Using X_1, X_2, R', and X_3, we can then *guess* C_1, C_2, C_5, and either $C_3 \cup A_4$ or $C_3 \cup C_4$. Now, we know A_2, A_3, A_5, and A_8. Note that for each position i in one of the four guessed sets C_1, C_2, C_5, and either $C_3 \cup A_4$ or $C_3 \cup C_4$, we can *guess* $t_{sol}[i]$ among only $|\Sigma| - 2$ choices.

For technical reasons, in our algorithm, we will not *guess* X_1, X_2, and X_3 separately. Rather, we will *guess* their union X and then split it into X_1, X_2, and X_3 by computing $X_1 = X \cap \{t_r \equiv s_u \not\equiv s_r\}$, $X_2 = X \cap \{s_r \equiv s_u \not\equiv t_r\}$, and $X_3 = \{t_r \not\equiv s_r \not\equiv s_u\}$. Similarly, in our algorithm, we will not *guess* C_1, C_2, and either $C_3 \cup A_4$ or $C_3 \cup C_4$ separately. Rather, we will *guess* their union Y and then split it by computing $Y \cap X_i$ for each $1 \leq i \leq 3$. We remind the reader that once we know X, Y, R', and C_5 (by *guessing*), we can use $t_r|_{P_r \cup R_u}$, $s_r|_{P_r \cup R_u}$, and $s_u|_{P_r \cup R_u}$ to compute $t_{sol}|_{(P_r \setminus Y) \cup (R_u \setminus C_5)}$ very easily. In contrast, we have to *guess* $t_{sol}[i]$ for each position $i \in Y \cup C_5$, and this is what our algorithm will do in Steps 10 and 14.3 of Subroutines 1 and 2 in Figures 6 and 7.

We next explain why we need to decide whether $|A_4| \leq |C_4|$ or not. By Lemma 5.1, the larger $\sum_{i=1}^{5} |C_i|$ is , the smaller $|B| = d(t_{sol}|_{[1..|t_{sol}|]\setminus(P_r \cup R_u)}, t_r|_{[1..|t_{sol}|]\setminus(P_r \cup R_u)})$ is (and so the shorter time

the 2-string algorithm will spend on computing $t_{sol}|_{[1..|t_{sol}|]\setminus(P_r\cup R_u)}$). So, we want our algorithm to take advantage of the sets C_1 through C_5. We also want to inherit the second main idea in the algorithm in Section 4, namely, obtaining $t_{sol}|_{P_r}$ from $s_u|_{P_r}$ by modifying $s_u|_{P'}$. Thus, it seems that we may first *guess* P', C_1, C_2, and C_3, and then obtain $t_{sol}|_{P_r}$ from s_u by modifying $s_u|_{P'}$ in such a way that $s_u[i]$ is changed to a letter different from both $s_r[i]$ and the old $s_u[i]$ for every $i \in C_1$, while $s_u[i]$ is changed to a letter different from both $t_r[i]$ and the old $s_u[i]$ for every $i \in C_2 \cup C_3$. However, in this way, we waste C_4 (because P' does not include C_4). To avoid wasting C_4, we have to look at A_4 and *guess* whether $|A_4| \leq |C_4|$. Basically, if $|A_4| \leq |C_4|$, we *guess* $C_3 \cup A_4$ instead of C_3; otherwise, we *guess* $C_3 \cup C_4$ instead of C_3.

When we call the 2-string algorithm to compute $t_{sol}|_{[1..|t_{sol}|]\setminus(P_r\cup R_u)}$, we already know $t_{sol}|_{P_r\cup R_u}$ and need to find an upper bound on $|B| = d(t_{sol}|_{[1..|t_{sol}|]\setminus(P_r\cup R_u)}, t_r|_{[1..|t_{sol}|]\setminus(P_r\cup R_u)})$ that is as small as possible. We can obtain one upper bound on $|B|$ from Lemma 5.1: If $|A_4| \leq |C_4|$, then $|B| \leq (3d - |P_r| - |R_u| - |P'| - |R'| - \sum_{i=1}^{5} |C_i|)/3 = (3d - |P_r| - |R_u| - |X| - |Y| + |A_4| - |C_4| - |R'| - |C_5|)/3$ because $|P'| = |X|$ and $\sum_{i=1}^{3} |C_i| = |Y| - |A_4|$; otherwise, $|B| \leq (3d - |P_r| - |R_u| - |P'| - |R'| - \sum_{i=1}^{5} |C_i|)/3 = (3d - |P_r| - |R_u| - |X| + |C_4| - |A_4| - |Y| - |R'| - |C_5|)/3$ because $|P'| = |X| - |C_4| + |A_4|$ and $\sum_{i=1}^{4} |C_i| = |Y|$. Moreover, we can obtain other upper bounds on $|B|$ from Lemma 3.1 as follows. We first mimic the computation of b' in Step 6 of the 2-string algorithm on input $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$ to obtain an upper bound b_1 on $d(t_r|_{[1..|t_r|]\setminus P_r}, t_{sol}|_{[1..|t_r|]\setminus P_r})$: If $|A_4| \leq |C_4|$, then $b_1 = \min\{d - k_1, k_1 - \ell_r - (|C_1| + |C_2| + |X_3 \cap Y| - |A_4| + |C_4|)\}$; otherwise, $b_1 = \min\{d - k_1, k_1 - \ell_r - (|C_1| + |C_2| + |X_3 \cap Y|)\}$, where $\ell_r = d(t_r, s_r) - d$ and $k_1 = d(t_r|_{P_r}, t_{sol}|_{P_r})$. We then mimic the computation of b' in Step 6 of the 2-string algorithm on input $\langle \mathcal{S} \setminus \{t_r, s_r\}, d; t_{sol}|_{P_r} + t_r|_{[1..|t_r|]\setminus P_r}, P_r, b_1\rangle$ to obtain an upper bound b_2 on $|B|$: $b_2 = \min\{b_1 - k_2, k_2 - \ell_2 - |C_5|\}$, where $k_2 = |R'|$, $\ell_2 = d(t_{sol}|_{P_r} + t_r|_{[1..|t_r|]\setminus P_r}, s_u) - d$. In our algorithm, b_1 and b_2 will be computed in Step 14.5.1 of Subroutines 1 and 2 in Figures 6 and 7.

Based on the above ideas, we design an algorithm for CSP as follows.

Theorem 5.2 *Let* $\beta = \alpha^2 + 1 - 2\alpha^{-1} + \alpha^{-2}$ *with* $\alpha = \sqrt[3]{\sqrt{|\Sigma| - 1} + 1}$. *Then, CSP can be solved in* $O\left(nL + dn^2 1.612^d \left(|\Sigma| + \beta^2 + \beta - 2\right)^d\right)$ *time.*

Proof. As in the proof of Theorem 4.4, we define the notations $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{T}_2$, and further assume that $\mathcal{A}_3$ does not halt in Step 2 and the string s_r found by $\mathcal{A}_3$ in Step 3 is the same as the string s found by $\mathcal{A}_2$ in Step 2 on input $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$.

First consider the case where a solution of the instance $\langle \mathcal{S} \setminus \{t_r\}, d; t_r, \emptyset, d\rangle$ is found at a child v of r in $\mathcal{T}_2$. Note that v is a leaf of $\mathcal{T}_2$ and the solution found at v is t_v. For this v, $\mathcal{A}_3$ will first guess an *arbitrary* string s_u (in Step 5), then correctly compute t_v (as t' in Step 11), and finally output t_v in Step 12. So, $\mathcal{A}_3$ is correct in this case.

Next consider a grandchild v of r in $\mathcal{T}_2$. It corresponds to an instance $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t_v, P_v, b_v\rangle$ of ECSP, where u is the parent of v in $\mathcal{T}_2$. $\mathcal{A}_3$ can correctly guess s_u in Step 5. Let t_{sol} be a solution

The 3-String Algorithm for the General Case

Input: An instance $(\mathcal{S}, d)$ of CSP.

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

1 − 5. Same as Steps 1 through 5 in the algorithm in Figure 3.

6. *Guess* a subset X of P_r with $X \leq d$. (*Comment:* If $|A_4| \leq |C_4|$, X is supposed to be P'. Otherwise, X is supposed to be $(P' \setminus A_4) \cup C_4$.)

7. *Guess* a subset Y of X with $|Y| \leq 3d - |P_r| - |X|$. (*Comment:* If $|A_4| \leq |C_4|$, Y is supposed to be $C_1 \cup C_2 \cup C_3 \cup A_4$. Otherwise, Y is supposed to be $\bigcup_{i=1}^{4} C_i$. In either case, Lemma 5.1 ensures that $|P_r| + |X| + |Y| \leq 3d$.)

8. Let $X_1 = \{s_u \equiv t_r \not\equiv s_r\} \cap X$, $X_2 = \{s_u \equiv s_r \not\equiv t_r\} \cap X$, $X_3 = \{t_r \not\equiv s_r \not\equiv s_u\} \cap X$, $C_1 = X_1 \cap Y$, $C_2 = X_2 \cap Y$, and $C_3' = X_3 \cap Y$. (*Comment:* $X_1 - C_1$, $X_2 - C_2$, and $X_3 - C_3'$ are supposed to be A_2, A_3, and A_5, respectively. So, if $|A_4| \leq |C_4|$, C_3' is supposed to be $C_3 \cup A_4$; otherwise, it is supposed to be $C_3 \cup C_4$.)

9. Run one of the subroutines in Figures 6 and 7. If it halts with a solution found, then output the solution and halt. Otherwise, run the other subroutine, output whatever it outputs, and halt. (*Comment:* Subroutine 1 handles the case where $|A_4| \leq |C_4|$, while Subroutine 2 handles the case where $|A_4| > |C_4|$.)

Figure 5: The 3-string algorithm for the general case.

of $(\mathcal{S}, d)$. For the four strings t_r, s_r, s_u, and t_{sol}, we refer to the notations defined in Lemma 5.1 and its proof (cf. Figure 4). We distinguish two cases depending on whether $a_4 \leq c_4$ or not.

Let us first consider the case where $a_4 \leq c_4$. Subroutine 1 works for this case. Basically, it guesses P' (as X) in Step 6 and $C_1 \cup C_2 \cup C_3 \cup A_4$ (as Y) in Step 7. In Step 8, $Y = C_1 \cup C_2 \cup C_3 \cup A_4$ is partitioned into three subsets C_1, C_2, $C_3 \cup A_4$ (as C_3'). These subsets are used to correctly compute $t_v|_{P_r}$ (as $t'|_{P_r}$ in Step 10). It further guesses R' and C_5 in Steps 14.1 and 14.2, respectively. The algorithm can now correctly compute t_v (as t in Step 14.4), P_v (trivially as $P_r \cup R_u$), and b_v (as b_2 in Step 14.5.1), and finally call $\mathcal{A}_2$ to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t_v, P_v, b_v \rangle$ in Step 14.5.2. Thus, in this case, if a solution of the instance $\langle \mathcal{S} - \setminus \{t_r\}, d; t_r, \emptyset, d \rangle$ is found at a descendant of v in $\mathcal{T}_2$, then $\mathcal{A}_3$ will find a solution, too.

Next consider the case where $a_4 > c_4$. Subroutine 2 works for this case. Basically, it guesses $(P' \setminus A_4) \cup C_4$ (as X) in Step 6 and $\bigcup_{i=1}^{4} C_i$ (as Y) in Step 7. In Step 8, $Y = \bigcup_{i=1}^{4} C_i$ is partitioned into three subsets C_1, C_2, $C_3 \cup C_4$ (as C_3'). These subsets are used to correctly compute $t_v|_{P_r}$ (as $t'|_{P_r}$ in Step 10). It further guesses R' and C_5 in Steps 14.1 and 14.2, respectively. The algorithm can now correctly compute t_v (as t in Step 14.4), P_v (trivially as $P_r \cup R_u$), and b_v (as b_2 in Step 14.5.1),

Subroutine 1

10. For each position $j \in C_1 \cup C_3'$, *guess* a letter $\hat{z}_j$ different from both $s_u[j]$ and $s_r[j]$; while for each position $j \in C_2$, *guess* a letter $\hat{z}_j$ different from both $s_u[j]$ and $t_r[j]$. Let the partial string $\hat{s}|_X$ be such that $\hat{s}|_X[j] = \hat{z}_j$ for all $j \in C_1 \cup C_2 \cup C_3'$, $\hat{s}|_X[j] = s_r[j]$ for all $j \in (X_1 - C_1) \cup (X_3 - C_3')$, and $\hat{s}|_X[j] = t_r[j]$ for all $j \in X_2 - C_2$.

11. Let $t' = \hat{s}|_X + s_u|_{P_r \setminus X} + t_r|_{[1..|t_r|] \setminus P_r}$. (*Comment:* $\hat{s}|_X + s_u|_{P_r \setminus X}$ is supposed to be $t_{sol}|_{P_r}$.)

12. If there is no $s \in \mathcal{S}$ with $d(t', s) > d$, then output t' and halt.

13. Let $c_4 = |\{t' \equiv s_u \not\equiv t_r \not\equiv s_r\}|$, $a_4 = |\{t' \equiv t_r \not\equiv s_r \not\equiv s_u\}|$, and $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. (*Comment:* By Lemma 5.1, $|R_u| \leq 3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i| = 3d - |P_r| - |X| - |Y| + a_4 - c_4$.)

14. If $d(t', t_r) \leq d$, $d(t', s_r) \leq d$, $d(t', s_u) > d$, $a_4 \leq c_4$, and $|R_u| \leq 3d - |P_r| - |X| - |Y| + a_4 - c_4$, then perform the following steps:

14.1. *Guess* a subset R' of R_u such that $|R'| \leq 3d - |P_r| - |R_u| - |X| - |Y| + a_4 - c_4$. (*Comment:* The upper bound on $|R'|$ used in this step follows from Lemma 5.1.)

14.2. *Guess* a subset C_5 of R' such that $|C_5| \leq 3d - |P_r| - |R_u| - |X| - |Y| + a_4 - c_4 - |R'|$. (*Comment:* The upper bound on $|C_5|$ used in this step follows from Lemma 5.1.)

14.3. For each position $j \in C_5$, *guess* a letter $\check{z}_j$ different from both $t_r[j]$ and $s_u[j]$. Let the partial string $\check{s}|_{R'}$ be such that $\check{s}|_{R'}[j] = \check{z}_j$ for all $j \in C_5$, and $\check{s}|_{R'}[j] = s_u[j]$ for all $j \in R' - C_5$.

14.4. Let $t = t'|_{P_r} + \check{s}|_{R'} + t_r|_{[1..|t_r|] \setminus (P_r \cup R')}$. (*Comment:* $\check{s}|_{R'} + t_r|_{R_u \setminus R'}$ is supposed to be $t_{sol}|_{R_u}$.)

14.5. If $d(t, t_r) \leq d$, $d(t, s_r) \leq d$, and $d(t, s_u) \leq d$, then perform the following steps:

14.5.1. Compute $\ell_r = d(t_r, s_r) - d$, $k_1 = d(t_r, t')$, $b_1 = \min\{d - k_1, k_1 - \ell_r - |C_1| - |C_2| - |C_3'| + a_4 - c_4\}$, $k_2 = |R'|$, $\ell_2 = d(t', s_u) - d$, and $b_2 = \min\{b_1 - k_2, k_2 - \ell_2 - |C_5|, (3d - |P_r| - |R_u| - |X| - |Y| + a_4 - c_4 - |R'| - |C_5|)/3\}$.

14.5.2. Call the 2-string algorithm to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t, P_r \cup R_u, b_2 \rangle$.

15. Return NULL.

Figure 6: The first subroutine called by the 3-string algorithm for the general case.

Subroutine 2

10. For each position $j \in C_1 \cup C_2 \cup C_3'$, *guess* a letter $\hat{z}_j$ different from both $t_r[j]$ and $s_r[j]$. Let the partial string $\hat{s}|_X$ be such that $\hat{s}|_X[j] = \hat{z}_j$ for all $j \in C_1 \cup C_2 \cup C_3'$, $\hat{s}|_X[j] = s_r[j]$ for all $j \in (X_1 - C_1) \cup (X_3 - C_3')$, and $\hat{s}|_X[j] = t_r[j]$ for all $j \in X_2 - C_2$.

11. Let $t' = \hat{s}|_X + s_u|_{\{s_u \equiv t_r \not\equiv s_r\} \setminus X_1} + s_u|_{\{s_u \equiv s_r \not\equiv t_r\} \setminus X_2} + t_r|_{\{s_u \not\equiv s_r \not\equiv t_r\} \setminus X_3} + t_r|_{[1..|t_r|] \setminus P_r}$. (*Comment:* $\hat{s}|_X + s_u|_{\{s_u \equiv t_r \not\equiv s_r\} \setminus X_1} + s_u|_{\{s_u \equiv s_r \not\equiv t_r\} \setminus X_2} + t_r|_{\{s_u \not\equiv s_r \not\equiv t_r\} \setminus X_3}$ is supposed to be $t_{sol}|_{P_r}$.)

12 – 13. Same as Steps 12 and 13 in Subroutine 1. (*Comment:* By Lemma 5.1, $|R_u| \leq 3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i| = 3d - |P_r| - |X| + c_4 - a_4 - |Y|$.)

14. If $d(t', t_r) \leq d$, $d(t', s_r) \leq d$, $d(t', s_u) > d$, $a_4 > c_4$, and $|R_u| \leq 3d - |P_r| - |X| + c_4 - a_4 - |Y|$, then perform the following steps:

14.1. *Guess* a subset R' of R_u such that $|R'| \leq 3d - |P_r| - |R_u| - |X| + c_4 - a_4 - |Y|$, where $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. (*Comment:* The upper bound on $|R'|$ used in this step follows from Lemma 5.1.)

14.2. *Guess* a subset C_5 of R' such that $|C_5| \leq 3d - |P_r| - |R_u| - |X| + c_4 - a_4 - |Y| - |R'|$. (*Comment:* The upper bound on $|C_5|$ used in this step follows from Lemma 5.1.)

14.3 – 14.4. Same as Steps 14.3 and 14.4 of Subroutine 1, respectively.

14.5. If $d(t, t_r) \leq d$, $d(t, s_r) \leq d$, and $d(t, s_u) \leq d$, then perform the following steps:

14.5.1. Compute $\ell_r = d(t_r, s_r) - d$, $k_1 = d(t_r, t')$, $b_1 = \min\{d - k_1, k_1 - \ell_r - |C_1| - |C_2| - |C_3'|\}$, $k_2 = |R'|$, $\ell_2 = d(t', s_u) - d$, and $b_2 = \min\{b_1 - k_2, k_2 - \ell_2 - |C_5|, (3d - |P_r| - |R_u| - |X| + c_4 - a_4 - |Y| - |R'| - |C_5|)/3\}$.

14.5.2. Call the 2-string algorithm to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t, P_r \cup R_u, b_2 \rangle$.

15. Return NULL.

Figure 7: The second subroutine called by the 3-string algorithm for the general case.

and finally call $\mathcal{A}_2$ to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t_v, P_v, b_v \rangle$ in Step 14.5.2. Thus, in this case, if a solution of the instance $\langle \mathcal{S} - \setminus \{t_r\}, d; t_r, \emptyset, d \rangle$ is found at a descendant of v in $\mathcal{T}_2$, then $\mathcal{A}_3$ will find a solution, too.

Next we estimate the time complexity of $\mathcal{A}_3$. The execution of $\mathcal{A}_3$ on input $(\mathcal{S}, d)$ can be modeled by a *search tree* $\mathcal{T}$ as follows. The root r of $\mathcal{T}$ corresponds to $(\mathcal{S}, d)$. For each quadruple $(s_u, X, Y, \langle \ldots, \hat{z}_j, \ldots \rangle)$ of possible outcomes of the guesses made in Steps 5, 6, 7, and 10 such that $\mathcal{A}_3$

does not execute Step 14.1, r has a child corresponding to the quadruple. Similarly, for each septuple $(s_u, X, Y, \langle \ldots, \hat{z}_j, \ldots \rangle, R', C_5, \langle \ldots, \check{z}_j, \ldots \rangle)$ of possible outcomes of the guesses made in Steps 5, 6, 7, 10, 14.1, 14.2, and 14.3 such that $\mathcal{A}_3$ executes Step 14.1, r has a child corresponding to the septuple. Moreover, for every child v corresponding to a septuple $(s_u, X, Y, \langle \ldots, \hat{z}_j, \ldots \rangle, R', C_5, \langle \ldots, \check{z}_j, \ldots \rangle)$ such that $\mathcal{A}_3$ executes Step 14.5.2, v has descendants in $\mathcal{T}$ so that the subtree of $\mathcal{T}$ rooted at v is the same as the search tree of $\mathcal{A}_2$ on input $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t, P_r \cup R_u, b_2 \rangle$.

Note that each non-leaf node of $\mathcal{T}$ has at least two children in $\mathcal{T}$. So, 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, let $\gamma = |\Sigma| - 2$, $h' = 3d - |P_r|$, and $f_{x,y} = h' - |R_u| - x - y$. Then, the number N of children of r in $\mathcal{T}$ that are not leaves of $\mathcal{T}$ satisfies the following inequality:

$$N \leq \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \tag{5.3}$$

where x, y, i, and j correspond to $|X|$, $|Y|$, $|R'|$, and $|C_5|$ in Subroutines 1 and 2, respectively.

Note that $|\Sigma| + 2\sqrt{|\Sigma| - 1} = \alpha^6$. So, by Eq. 5.3 and Lemma 3.3, the number N' of leaves at depth 2 or more in $\mathcal{T}$ satisfies the following inequalities:

$$N' \leq \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \binom{d - b_2}{b_2} \alpha^{6b_2}.$$

So, by Lemma 4.3,

$$N' \leq 1.612^d \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \alpha^{6b_2}. \tag{5.4}$$

By Step 14.5.1, $b_1 \leq 0.5d$, $b_2 \leq 0.5b_1 \leq 0.25d$, and $3b_2 \leq h' - |R_u| - x - y - i - j$. Thus, we have

$$\begin{aligned}
N' &\leq 1.612^d \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \alpha^{2(f_{x,y}-i-j)} && (5.5) \\
&\leq 1.612^d \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \alpha^{2f_{x,y}} \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \frac{1}{\alpha^{2i}} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \left(\frac{\gamma}{\alpha^2}\right)^j \\
&\leq 1.612^d \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \alpha^{2f_{x,y}} \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \frac{1}{\alpha^{2i}} \left(1 + \frac{\gamma}{\alpha^2}\right)^i \\
&\leq 1.612^d \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \alpha^{2f_{x,y}} \left(1 + \frac{1}{\alpha^2} + \frac{\gamma}{\alpha^4}\right)^{|R_u|} \\
&\leq 1.612^d \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \frac{1}{\alpha^{2x}} \sum_{y=0}^{h'-x} \binom{x}{y} \left(\frac{\gamma}{\alpha^2}\right)^y \alpha^{2h'} \left(\frac{1}{\alpha^2} + \frac{1}{\alpha^4} + \frac{\gamma}{\alpha^6}\right)^{|R_u|} && (5.6)
\end{aligned}$$

By Step 14, $|R_u| \leq 3d - |P_r| - x - y = h' - x - y$. Thus, by Eq. 5.6, we have

$$\begin{aligned} N' &\leq 1.612^d \sum_{s_u} \beta^{h'} \sum_{x=0}^{d} \binom{|P_r|}{x} \frac{1}{\beta^x} \sum_{y=0}^{h'-x} \binom{x}{y} \left(\frac{\gamma}{\beta}\right)^y \quad (5.7) \\ &\leq 1.612^d \sum_{s_u} \beta^{h'} \sum_{x=0}^{d} \binom{|P_r|}{x} \frac{1}{\beta^x} \left(1 + \frac{\gamma}{\beta}\right)^x \\ &\leq 1.612^d \sum_{s_u} \beta^{h'} \left(1 + \frac{1}{\beta} + \frac{\gamma}{\beta^2}\right)^{|P_r|}. \quad (5.8) \end{aligned}$$

One can verify that $\beta \geq 1 + \frac{1}{\beta} + \frac{\gamma}{\beta^2}$ for all alphabets Σ with $|\Sigma| \geq 2$. Now, since $h' + |P_r| = 3d$ and $|P_r| > d$, Eq. 5.8 implies the following:

$$N' \leq 1.612^d \sum_{s_u} \beta^{2d} \left(1 + \frac{1}{\beta} + \frac{\gamma}{\beta^2}\right)^d \leq (n-2)1.612^d \left(\beta^2 + \beta + \gamma\right)^d. \quad (5.9)$$

We next bound the number of leaves at depth 1 in $\mathcal{T}$. Clearly, the number M_1 of leaves at depth 1 in $\mathcal{T}$ corresponding to a quadruple satisfies the following inequalities:

$$M_1 \leq \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \leq \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \beta^{h'-x-y}. \quad (5.10)$$

Note that the right-hand sides of Eq. 5.10 and 5.7 are very similar. Consequently, as we used the bound on N' in Eq. 5.7 to obtain the bound on N' in Eq. 5.9, we can prove

$$M_1 \leq (n-2)\left(\beta^2 + \beta + \gamma\right)^d. \quad (5.11)$$

On the other hand, the number M_2 of leaves at depth 1 in $\mathcal{T}$ corresponding to a septuple satisfies the following inequalities:

$$\begin{aligned} M_2 &\leq \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \\ &\leq \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \alpha^{6b_2}, \end{aligned}$$

where we simply let $b_2 = (f_{x,y} - i - j)/3$ to ensure that $\alpha^{6b_2} \geq 1$. Consequently, as we used the bound on N' in Eq. 5.4 to obtain the bound on N' in Eq. 5.9, we can prove

$$M_2 \leq (n-2)\left(\beta^2 + \beta + \gamma\right)^d. \quad (5.12)$$

Therefore, by Eq. 5.4, 5.11, and 5.12, the total number of leaves in $\mathcal{T}$ is

$$N' + M_1 + M_2 = O\left((n-2)1.612^d \left(\beta^2 + \beta + \gamma\right)^d\right).$$

14. If $d(t', t_r) \leq d$, $d(t', s_r) \leq d$, and $a_4 \leq c_4$, then perform the following steps:

14.1. If $d(t', s_u) \leq d$, then select a string $\tilde{s}_u \in \mathcal{S} \setminus \{t_r, s_r, s_u\}$ such that $d(t', \tilde{s}_u) > d$ and $|\{t_r \equiv s_r \not\equiv \tilde{s}_u\}| \leq |\{t_r \equiv s_r \not\equiv s\}|$ for all $s \in \mathcal{S} \setminus \{t_r, s_r, s_u\}$ with $d(t', s) > d$; further let s_u refer to the same string as $\tilde{s}_u$ does, and then compute $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. (*Comment:* Since $\max\{d(t', t_r), d(t', s_r), d(t', s_u)\} \leq d$ but t' is not a solution, $\tilde{s}_u$ must exist.)

14.2. If $|R_u| \leq 3d - |P_r| - |X| - |Y| + a_4 - c_4$, then perform the following steps:

14.2.1 − 14.2.5. Same as Steps 14.1 through 14.5 of Subroutine 1 in Figure 6, respectively.

Figure 8: The modification of Step 14 of Subroutine 1.

14. If $d(t', t_r) \leq d$, $d(t', s_r) \leq d$, and $a_4 > c_4$, then perform the following steps:

14.1. Same as Step 14.1 in Figure 8.

14.2. If $|R_u| \leq 3d - |P_r| - |X| + c_4 - a_4 - |Y|$, then perform the following steps:

14.2.1 − 14.2.5. Same as Steps 14.1 through 14.5 of Subroutine 2 in Figure 7, respectively.

Figure 9: The modification of Step 14 of Subroutine 2.

Consequently, $\mathcal{A}_3$ runs in time

$$O\left(nL + n^2 d 1.612^d \left(\beta^2 + \beta + \gamma\right)^d\right),$$

because each node of $\mathcal{T}$ takes $O(nd)$ time. □

As in the binary case (cf. Subsection 4.2), we can improve the time bound of the 3-string algorithm by a factor of n. Again, the crux is to modify Step 5 as in Subsection 4.2. Accordingly, we need to replace Step 14 of Subroutines 1 and 2 as in Figures 8 and 9, respectively.

The key point here is the following lemma (which is very similar to Lemma 4.5):

Lemma 5.3 *Consider the time point where the refined algorithm just selected $\tilde{s}_u$ in Step 14.1 in Figure 8 or 9 but has not let s_u refer to the same string as $\tilde{s}_u$ does. If $a_4 \leq c_4$, then $|P_r| + |X| + |Y| - a_4 + c_4 + |\tilde{R}_u| + |\tilde{R}'| \leq 3d - 3|\tilde{B}| - |\tilde{C}_5|$; otherwise, $|P_r| + |X| - c_4 + a_4 + |Y| + |\tilde{R}_u| + |\tilde{R}'| \leq 3d - 3|\tilde{B}| - |\tilde{C}_5|$, where $\tilde{R}_u = \{t_r \equiv s_r \not\equiv \tilde{s}_u\}$, $\tilde{R}' = \{t_{sol} \not\equiv t_r\} \cap \tilde{R}_u$, $\tilde{B} = \{t_r \equiv s_r \equiv \tilde{s}_u \not\equiv t_{sol}\}$, and $\tilde{C}_5 = \{t_r \equiv s_r \not\equiv \tilde{s}_u \not\equiv t_{sol}\}$.*

Proof. Let $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. By the modified Step 5, $|\tilde{R}_u| \leq |R_u|$. Since $d(t', s_u) = |P'| + |R_u| \leq d$ and $d(t', \tilde{s}_u) = d(t'|_{P_r}, \tilde{s}_u|_{P_r}) + |\tilde{R}_u| > d$, we have $|P'| < d(t'|_{P_r}, \tilde{s}_u|_{P_r})$. Because

$d(t_{sol}|_{P_r}, s_u|_{P_r}) = d(t'|_{P_r}, s_u|_{P_r}) = |P'|$, it follows that $d(t_{sol}|_{P_r}, s_u|_{P_r}) < d(t'|_{P_r}, \tilde{s}_u|_{P_r})$. Consequently, $d(t_{sol}|_{P_r}, s_u|_{P_r}) < d(t_{sol}|_{P_r}, \tilde{s}_u|_{P_r})$ for $t'|_{P_r} = t_{sol}|_{P_r}$.

Consider the string $\hat{s}_u = s_u|_{P_r} + \tilde{s}_u|_{[1..|s_u|]\setminus P_r}$. Note that $d(t_{sol}, \hat{s}_u) - d(t_{sol}, \tilde{s}_u) = d(t_{sol}|_{P_r}, s_u|_{P_r}) - d(t_{sol}|_{P_r}, \tilde{s}_u|_{P_r})$. So, by the last inequality in the last paragraph, $d(t_{sol}, \hat{s}_u) < d(t_{sol}, \tilde{s}_u)$. Consequently, $d(t_{sol}, \hat{s}_u) \leq d$ for $d(t_{sol}, \tilde{s}_u) \leq d$. Now, applying Lemma 5.1 with s_u there being replaced by $\hat{s}_u$ here, we have $|P_r| + |P'| + |\tilde{R}_u| + |\tilde{R}'| \leq 3d - 3|\tilde{B}| - \sum_{i=1}^{4} |C_i| - |\tilde{C}_5|$. Thus, $|P_r| + |P'| + \sum_{i=1}^{4} |C_i| + |\tilde{R}_u| + |\tilde{R}'| \leq 3d - 3|\tilde{B}| - |\tilde{C}_5|$. If $a_4 \leq c_4$, then by the algorithm, $|X| = |P'|$ and $|Y| = \sum_{i=1}^{3} |C_i| + a_4$, implying that $|P_r| + |X| + |Y| - a_4 + c_4 + |\tilde{R}_u| + |\tilde{R}'| \leq 3d - 3|\tilde{B}| - |\tilde{C}_5|$. On the other hand, if $a_4 > c_4$, then by the algorithm, $|X| = |P'| - a_4 + c_4$ and $|Y| = \sum_{i=1}^{4} |C_i|$, implying that $|P_r| + |X| - c_4 + a_4 + |Y| + |\tilde{R}_u| + |\tilde{R}'| \leq 3d - 3|\tilde{B}| - |\tilde{C}_5|$. □

Theorem 5.4 *The refined 3-string algorithm solves CSP in* $O\left(nL + dn1.612^d \left(|\Sigma| + \beta^2 + \beta - 2\right)^d\right)$ *time.*

Proof. To see the correctness of the refined algorithm, it suffices to consider the case where $\tilde{s}_u$ is selected in Step 14.1 in Figure 8 (respectively, Figure 9). In this case, by Lemma 5.3, the algorithm can correctly guess $\tilde{R}'$ and $\tilde{C}_5$ in Steps 14.1 and 14.2 of Subroutine 1 (respectively, Subroutine 2) in Figure 6 (respectively, Figure 7). The correctness of the computation of the upper bound b_2 on the remaining distance between t and t_{sol} in Step 14.3.1 of Subroutine 1 (respectively, Subroutine 2) in Figure 6 (respectively, Figure 7) again follows from Lemma 5.3. From these facts, it is not hard to see that the refined algorithm is correct.

Since the refined algorithm does not guess s_u, we can mimic the analysis in the proof of Theorem 5.2 to show that the refined algorithm runs in $O\left(nL + dn1.612^d \left(|\Sigma| + \beta^2 + \beta - 2\right)^d\right)$ time. □

When $|\Sigma| = 4$, the time bound of the 3-string algorithm given in Theorem 5.2 is $O\left(nL + nd13.183^d\right)$ which is better than the previously best time bound $O\left(nL + nd13.922^d\right)$ in [2]. However, when $|\Sigma| \geq 7$, the time bound of the 3-string algorithm given in Theorem 5.2 is (slightly) worse than that in [2] (but better than that in [27]).

Acknowledgments

Zhi-Zhong Chen was supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 20500021. Bin Ma was supported in part by China 863 National High-tech R&D Program (2008AA02Z313), NSERC (RGPIN 238748-2006) and a start up grant at University of Waterloo. Lusheng Wang was fully supported by a grant from City University of Hong Kong (project no. 7002452).

References

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

[2] Z. Z. Chen and L. Wang. Fast exact algorithms for the closest string and substring problems with application to the planted (l, d)-motif model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2009. Submitted for publication.

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

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

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

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

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

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

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

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

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

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

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

[14] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string search problems. In *SODA 1999*, pages 633–642, 1999.

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

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

[17] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. In *Proceedings of the 12th Annual International Conference on Computational Biology (RECOMB)*, pages 396–409, 2008.

[18] D. Marx. The closest substring problem with small distances. In *FOCS 2005*, pages 63–72, 2005.

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

[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 Journal on Computing*, 2004.

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

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

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

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

[25] L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. In *The Third International Frontiers of Algorithmics Workshop*, pages 261–270, 2009.

[26] 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.

[27] R. Zhao and N. Zhang. A more efficient closest string algorithm. In *2nd International Conference on Bioinformatics and Computational Biology*, 2010. To appear.