Journal of Computer and System Sciences 78 (2012) 164–178

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A three-string approach to the closest string problem

Zhi-Zhong Chen[a,*], Bin Ma[b,1], Lusheng Wang[c,2]

[a] Department of Mathematical Sciences, Tokyo Denki University, Ishizaka, Hatoyama, Hiki, Saitama, 359-0394, Japan
[b] School of Computer Science, University of Waterloo, 200 University Ave. W, Waterloo, ON, Canada N2L3G1
[c] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 18 May 2010
Received in revised form 11 January 2011
Accepted 21 January 2011
Available online 28 January 2011

Keywords:
Computational biology
The closest string problem
Fixed-parameter algorithms

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 $O(nL + nd^3 \cdot 6.731^d)$ time, while the previous best runs in $O(nL + nd \cdot 8^d)$ time. We then extend the algorithm to arbitrary alphabet sizes, obtaining an algorithm that runs in time $O(nL + nd \cdot (1.612(|\Sigma| + \beta^2 + \beta - 2))^d)$, 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).

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 $d(t_{sol}, s) \leqslant d$ 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 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 [18,17]. 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 $O(L \cdot n^{O(\epsilon^{-2})})$.

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

* Corresponding author. Fax: +81 49 296 7072.
E-mail addresses: zzchen@mail.dendai.ac.jp (Z.-Z. Chen), binma@uwaterloo.ca (B. Ma), lwang@cs.cityu.edu.hk (L. Wang).
[1] Fax: +1 519 885 1208.
[2] Fax: +852 2788 8614.

0022-0000/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2011.01.003

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 $O(nL + n \cdot (d+1)^{d+1})$. Ma and Sun [17] gave an algorithm that runs in $O(nL + nd \cdot (16(|\Sigma| - 1))^d)$ time; their algorithm 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 $O(nL + nd \cdot (2^{3.25}(|\Sigma| - 1))^d)$. Chen and Wang [2] further improved the time complexity to $O(nL + nd \cdot 8^d)$ for binary strings and to $O(nL + nd \cdot (\sqrt{2}|\Sigma| + \sqrt[4]{8}(\sqrt{2}+1)(1+\sqrt{|\Sigma|-1}) - 2\sqrt{2})^d)$ for non-binary strings. Independently, Zhao and Zhang [27] provided an algorithm with running time $O(nL + nd \cdot (2|\Sigma| + 4\sqrt{|\Sigma|-1})^d)$. Note that the algorithm in [2] is as fast as the algorithm in [27] for binary strings but is faster for other 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 $O(nL + nd^3 \cdot 6.731^d)$ time, while the previous best runs in $O(nL + nd \cdot 8^d)$ time. We then extend the algorithm to arbitrary strings, obtaining an algorithm that runs in time $O(nL + nd \cdot (1.612(|\Sigma| + \beta^2 + \beta - 2))^d)$, where $\beta = \alpha^2 + 1 - 2\alpha^{-1} + \alpha^{-2}$ with $\alpha = \sqrt[3]{\sqrt{|\Sigma|-1}+1}$. Note that β is roughly $\sqrt[3]{|\Sigma| + 2\sqrt{|\Sigma|-1}}$. In particular, in the very important case where $|\Sigma| = 4$ (i.e., the case of DNA strings), our algorithm runs in $O(nL + nd \cdot 13.183^d)$ time, while the previous best runs in $O(nL + nd \cdot 13.921^d)$ 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 $1 \leqslant i < j \leqslant h$, 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 $1 \leqslant i < j \leqslant h$. 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 $h \geqslant 2$, $k \geqslant 1$, and $\ell \geqslant 0$, $\{s_1 \equiv s_2 \equiv \cdots \equiv s_h \not\equiv t_1 \not\equiv t_2 \not\equiv \cdots \not\equiv t_k \equiv u_1 \equiv u_2 \equiv \cdots \equiv u_\ell\}$ 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 $s_1, s_2, \ldots, s_n$ 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 $d(t, t_{sol}) \leqslant d$. Using some strategies, the algorithm

```
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 $Y \subseteq X \subseteq R$, $\ell \leqslant |X| \leqslant b$, and $|Y| \leqslant |X| - \ell$. (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 $b' = \min\{b - |X|, |X| - \ell - |Y|\}$. (Comment: $d(t,t') = |X|$.)
  7. Solve $\langle \mathcal{S} \setminus \{s\}, d; t', P \cup R, b'\rangle$ recursively.
  8. Return NULL.
```

Fig. 1. The algorithm given in [2].

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 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. $d(t_{sol}, t) \leqslant b$.
3. For every string $s \in \mathcal{S}$, $d(t_{sol}, s) \leqslant d$.

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 [12,17,2] 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 $d(s, t_{sol}) \leqslant d$, 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 $c \leqslant k$ 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 Fig. 1.

Lemma 3.1. *(See [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, $b' \leqslant b - k$ and $b' \leqslant k - \ell - c$. Consequently, $b' \leqslant \frac{b-\ell-c}{2}$.*

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]:

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

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

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

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

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

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 $u_1, u_2, \dots, u_{h-1}$ 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 $0 \leqslant i \leqslant h-1$, 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, $k_0 \geqslant \ell_r + b_{u_1}$ and $b_{u_i} \geqslant k_i + b_{u_{i+1}}$ for each $1 \leqslant i \leqslant h-1$. So, $k_0 \geqslant \ell_r + \sum_{i=1}^{h-1} k_i + b_u$. Again, by the computation of b' in Step 6 of the 2-string algorithm, $b_{u_i} \geqslant 2b_{u_{i+1}}$ and $k_i \geqslant b_{u_{i+1}}$ for each $0 \leqslant i \leqslant h-1$. So, for each $0 \leqslant i \leqslant h-1$, $b_{u_i} \geqslant 2^{h-i} b_u$ and $k_i \geqslant 2^{h-i-1} b_u$. Now, $d(t_r, t_u) = \sum_{i=0}^{h-1} k_i \geqslant \ell_r + b_u + 2\sum_{i=1}^{h-1} k_i \geqslant \ell_r + b_u + 2b_u \sum_{i=1}^{h-1} 2^{h-i-1} = \ell_r + (2^h - 1)b_u$. Thus, by Lemma 3.2, $T(d, b_u) \leqslant \binom{d-(2^{h-1}-1)b_u}{b_u} \cdot (|\Sigma| + 2\sqrt{|\Sigma|-1})^{b_u}$. $\square$

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 $|\mathcal{S}| \geqslant 3$.

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 $R_u \setminus P_r = \{t_r \equiv s_r \not\equiv s_u\}$.

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 Section 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, $|P_r| + |P'| + |R_u| + |R'| \leqslant 3d - 3|B|$.*

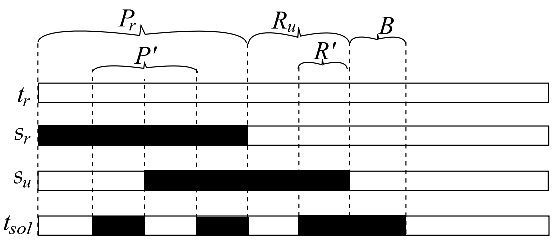


Fig. 2. Strings t_r, s_r, s_u, and t_{sol} in Lemma 4.1, where for each position $i \in [1..|t_r|]$, two of the strings have the same letter at the ith position if and only if the two strings are illustrated in the same color at the ith position.

```
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 $d(t_r, s_r) \leqslant d(t_r, t_{sol}) + d(t_{sol}, s_r) \leqslant d + d = 2d$.)
  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 $|P'| \leqslant d$.
  7. Let $t' = \bar{s}_u|_{P'} + s_u|_{P_r \setminus P'} + t_r|_{[1..|t_r|] \setminus P_r}$. (Comment: $t'|_{P_r}$ is supposed to be $t_{sol}|_{P_r}$.)
  8. If there is no $s \in \mathcal{S}$ with $d(t', s) > d$, then output $t'$ and halt.
  9. If $d(t', t_r) \leqslant d$, $d(t', s_r) \leqslant d$, and $d(t', s_u) > d$, then perform the following steps:
    9.1. Guess a subset $R'$ of $R_u$ such that $|R'| \leqslant 3d - |P_r| - |R_u| - |P'|$, 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 $t = t'|_{P_r} + \bar{t}_r|_{R'} + t_r|_{[1..|t_r|] \setminus (P_r \cup R')}$. (Comment: $t|_{P_r \cup R_u}$ is supposed to be $t_{sol}|_{P_r \cup R_u}$.)
    9.3. If $d(t, t_r) \leqslant d$, $d(t, s_r) \leqslant d$, and $d(t, s_u) \leqslant d$, 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 $b_2 \leqslant \min\{b_1 - k_2, k_2 - \ell_2\}$ 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, $b_2 \leqslant (3d - |P_r| - |R_u| - |P'| - |R'|)/3$ 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.
```

Fig. 3. The 3-string algorithm for the binary case.

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

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

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. Fig. 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 $(|P_r| + |P'|) + (|R_u| + |R'|) + 3|B| \leqslant 3d$. The lemma is proved. □

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 $|P_r| + |P'| + |R_u| + |R'| \leqslant 3d - 3|B|$, 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 Fig. 3.

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

Proposition 4.2.

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

Proof. The proposition is clearly true when $k = 0$ or p. Consider an arbitrary integer k with $1 \leqslant k \leqslant p-1$. Let $\alpha = \frac{k}{p}$. By Stirling's formula, $\binom{p}{k} \leqslant \alpha^{-\alpha p}(1-\alpha)^{-(1-\alpha)p}$. So, to finish the proof, it suffices to show that $\alpha^{-\alpha}(1-\alpha)^{-(1-\alpha)} \leqslant (\frac{1+\sqrt{5}}{2})^{1+\alpha}$. Taking the logarithms of both sides of this inequality, we have that $-\alpha\log\alpha - (1-\alpha)\log(1-\alpha) \leqslant (1+\alpha)\log\frac{1+\sqrt{5}}{2}$. 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 $f(\alpha) \geqslant 0$. 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 $f(\frac{2}{3+\sqrt{5}}) \geqslant 0$. Since the last inequality can be easily verified, the proposition is proved. $\square$

The next proposition slightly strengthens Proposition 4.2 when $p \geqslant 3k$.

Proposition 4.3. *Suppose that $p \geqslant 3k$. Then,*

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

Proof. The proof is similar to that of Proposition 4.2. What we need to prove here is that $\alpha^{-\alpha}(1-\alpha)^{-(1-\alpha)} \leqslant (\sqrt[4]{6.75})^{1+\alpha}$. 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 $f(\alpha) \geqslant 0$. 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 $\frac{1}{1+\sqrt[4]{6.75}} > \frac{1}{3} \geqslant \alpha$ for $p \geqslant 3k$. So, when $p \geqslant 3k$, $f(\alpha)$ achieves the minimum value at $\alpha = \frac{1}{3}$. Thus, it remains to prove that $f(\frac{1}{3}) \geqslant 0$. Since the last inequality can be easily verified, the proposition is proved. $\square$

Theorem 4.4. *The binary case of CSP can be solved in $O(nL + n^2d^2 \cdot 6.731^d)$ 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 $d \leqslant 25$, then clearly $\mathcal{A}_3$ runs in $O(n^2)$ time and the theorem is true. So, we further assume that $d \geqslant 26$.

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:

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

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

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

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

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

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

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

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:

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

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

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

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

where we simply let $b_2 = h - x$ to ensure that $\binom{d-b_2}{b_2}4^{b_2} \geqslant 1$. 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 $M_2 \leqslant 2nd \cdot 6.731^d$. Therefore, by Eqs. (4.6) and (4.7), the total number of leaves in $\mathcal{T}$ is $N' + M_1 + M_2 \leqslant 6nd \cdot 6.731^d$. Consequently, $\mathcal{A}_3$ runs in $O(nL + n^2d^2 \cdot 6.731^d)$ 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 Section 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, which is needed to ensure the correct use of the 2-string algorithm in Step 9.3.2. Thus, if $d(t', s_u) \leqslant d$, 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 Section 4.1 is replaced by the following:

```
9. If $d(t', t_r) \leqslant d$ and $d(t', s_r) \leqslant d$, then perform the following steps:
        9.0. If $d(t', s_u) \leqslant d$, 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 $\max\{d(t', t_r), d(t', s_r), d(t', s_u)\} \leqslant d$ but $t'$
             is not a solution, $\tilde{s}_u$ must exist.)
    9.1–9.3. Same as those of the algorithm in Fig. 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, $|P_r| + |P'| + |\tilde{R}_u| + |R'| \leqslant 3d - 3|\tilde{B}|$, 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 $\tilde{B} = \{t_r \equiv s_r \equiv \tilde{s}_u \not\equiv t_{sol}\}$.*

```
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 Fig. 3.
  6. Compute $t_{sol}|_{P_r}$, where $t_{sol}$ is an arbitrary targeted solution to $(\mathcal{S}, d)$.
  7. Let $t' = t_{sol}|_{P_r} + t_r|_{[1..|t_r|]\setminus P_r}$.
  8. If there is no $s \in \mathcal{S}$ with $d(t', s) > d$, then output $t'$ and halt.
  9. Check whether $d(t', t_r) \leqslant d$, $d(t', s_r) \leqslant d$, and $d(t', s_u) > d$. If the checking fails, return NULL.
 10. Compute $t_{sol}|_{R_u}$, where $R_u = \{t_r \equiv s_r \not\equiv s_u\}$.
 11. Construct a string $t = t_{sol}|_{P_r} + t_{sol}|_{R_u} + t_r|_{[1..|t_r|]\setminus(P_r \cup R_u)}$.
 12. Check whether $d(t, t_r) \leqslant d$, $d(t, s_r) \leqslant d$, and $d(t, s_u) \leqslant d$. If the checking fails, return NULL.
 13. Compute an upper bound $b_2$ on the size of $B = \{j \in [1..|t_r|] \setminus (P_r \cup R_u) \mid t[j] \neq t_{sol}[j]\}$.
 14. 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. If the 2-string algorithm returns a solution, then return it; otherwise, return NULL.
```

Fig. 4. The outline of the 3-string algorithm for the general case.

Proof. Let $R_u = \{t_r \equiv s_r \not\equiv s_u\}$. By the modified Step 5, $|\tilde{R}_u| \leqslant |R_u|$. Since $d(t', s_u) = |P'| + |R_u| \leqslant d$ 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, $|P_r| + d(t'|_{P_r}, \tilde{s}_u|_{P_r}) + |\tilde{R}_u| + |R'| \leqslant 3d - 3|\tilde{B}|$. Since $|P'| < d(t'|_{P_r}, \tilde{s}_u|_{P_r})$, we have $|P_r| + |P'| + |\tilde{R}_u| + |R'| \leqslant 3d - 3|\tilde{B}|$. □

Theorem 4.6. *The refined 3-string algorithm solves the binary case of CSP in $O(nL + nd^3 \cdot 6.731^d)$ 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 $|R'| \leqslant 3d - |P_r| - |P'| - |\tilde{R}_u|$. 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 Section 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 Section 4.1. Indeed, instead of Eq. (4.4) in Section 4.1, we only have the following inequalities for the refined algorithm:

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

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 $O(nL + nd^3 \cdot 6.731^d)$ time. □

The previously best time complexity for the binary case is $O(nL + nd \cdot 8^d)$ [2,27]. The new algorithm is better when d is large (say, $\geqslant 44$). When d is small, the algorithm in the next section is better because its running time for binary strings is $O(nL + nd \cdot 6.911^d)$.

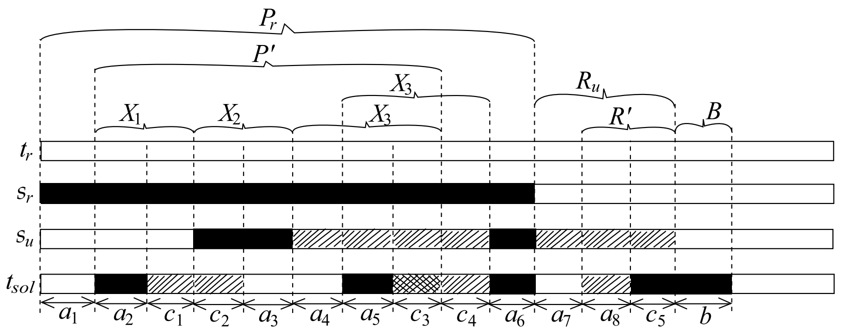
5. Extension to arbitrary alphabets

In this section, we extend the algorithm in Section 4 so that it works for arbitrary alphabets. The outline of the extended algorithm is the same as that of the algorithm in Section 4 and is shown in Fig. 4.

Note that the outline in Fig. 4 lacks three details, namely, the computations of $t_{sol}|_{P_r}$, $t_{sol}|_{R_u}$, and b_2. The remainder of this section is devoted to the details.

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 $d(t_{sol}, t_r) \leqslant d$, $d(t_{sol}, s_r) \leqslant d$, and $d(t_{sol}, s_u) \leqslant d$. Let $P_r = \{t_r \not\equiv s_r\}$, $R_u = \{t_r \equiv s_r \not\equiv s_u\}$, $P' = \{t_{sol} \not\equiv s_u\} \cap P_r$, $R' = \{t_{sol} \not\equiv t_r\} \cap R_u$, $B = \{t_r \equiv s_r \equiv s_u \not\equiv t_{sol}\}$, $C_1 = \{s_u \equiv t_r \not\equiv s_r \not\equiv t_{sol}\}$, $C_2 = \{s_u \equiv s_r \not\equiv t_r \not\equiv t_{sol}\}$, $C_3 = \{t_r \not\equiv s_r \not\equiv s_u \not\equiv t_{sol}\}$, $C_4 = \{s_u \equiv t_{sol} \not\equiv t_r \not\equiv s_r\}$,*



P_r
P'
X_3
R_u
X_1
X_2
X_3
R'
B
t_r
s_r
s_u
t_{sol}
a_1 a_2 c_1 c_2 a_3 a_4 a_5 c_3 c_4 a_6 a_7 a_8 c_5 b

Fig. 5. 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.

and $C_5 = \{t_r \equiv s_r \not\equiv s_u \not\equiv t_{sol}\}$. *Then,*

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

Proof. Let $b = |B|$ and $c_i = |C_i|$ for all $i \in \{1, \ldots, 5\}$ (cf. Fig. 5). 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 $d(t_{sol}, t_r) \leqslant d$, $d(t_{sol}, s_r) \leqslant d$, and $d(t_{sol}, s_u) \leqslant d$,

$$d(t_r, t_{sol}) = a_2 + a_5 + a_6 + a_8 + b + \sum_{i=1}^{5} c_i \leqslant d,$$

$$d(s_r, t_{sol}) = a_1 + a_3 + a_4 + a_8 + b + \sum_{i=1}^{5} c_i \leqslant d,$$

$$d(s_u, t_{sol}) = c_1 + c_2 + c_3 + c_5 + b + a_7 + \sum_{i=2}^{5} a_i \leqslant d.$$

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

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

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 Eqs. (5.1) and (5.2), the lemma holds. □

To understand the remainder of this section, Fig. 5 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 $|A_4| \leqslant |C_4|$, then $X_3 = A_4 \cup A_5 \cup C_3$; otherwise, $X_3 = A_5 \cup C_3 \cup C_4$.

```
(1) Guess a subset $X$ of $P_r$ with $|X| \leqslant d$. (Comment: $X$ is supposed to be $P'$.)
(2) Guess a subset $Y$ of $X$ with $|Y| \leqslant 3d - |P_r| - |X|$. (Comment: $Y$ is supposed to be $C_1 \cup C_2 \cup C_3 \cup A_4$. Note
    that by Lemma 5.1, $|P_r| + |P'| + \sum_{i=1}^{3} |C_i| + |A_4| \leqslant 3d$ because $|A_4| \leqslant |C_4|$.)
(3) Compute $X_1 = \{s_u \equiv t_r \not\equiv s_r\} \cap X$, $X_2 = \{s_u \equiv s_r \not\equiv t_r\} \cap X$, $X_3 = \{t_r \not\equiv s_r \not\equiv s_u\} \cap X$, $C_1 = X_1 \cap Y$, $C_2 = X_2 \cap Y$,
    $C_3' = X_3 \cap Y$, $C_4 = \{t_r \not\equiv s_r \not\equiv s_j\} \setminus X_3$, $A_1 = \{s_u \equiv t_r \not\equiv s_r\} \setminus X_1$, $A_2 = X_1 \setminus C_1$, $A_3 = X_2 \setminus C_2$, $A_5 = X_3 \setminus C_3'$,
    and $A_6 = P_r \setminus (X \cup A_1 \cup C_4)$. (Comment: $C_3'$ is supposed to be $C_3 \cup A_4$.)
(4) Compute $t_{sol}|_{P_r}$ as follows.
    (4.1) For each position $j \in C_1 \cup C_3'$, guess $t_{sol}[j] \in \Sigma \setminus \{s_u[j], s_r[j]\}$.
    (4.2) For each position $j \in C_2$, guess $t_{sol}[j] \in \Sigma \setminus \{s_u[j], t_r[j]\}$.
    (4.3) For each $j \in A_2 \cup A_5$, let $t_{sol}[j] = s_r[j]$.
    (4.4) For each $j \in A_3$, let $t_{sol}[j] = t_r[j]$.
    (4.5) For each $j \in A_1 \cup C_4 \cup A_6$, let $t_{sol}[j] = s_u[j]$.
```

Fig. 6. Computing $t_{sol}|_{P_r}$ when $|A_4| \leqslant |C_4|$.

```
(1) Guess a subset $X$ of $P_r$ with $|X| \leqslant d$. (Comment: $X$ is supposed to be $(P' \setminus A_4) \cup C_4$. Note that $|X| < |P'|$
    because $|A_4| > |C_4|$.)
(2) Guess a subset $Y$ of $X$ with $|Y| \leqslant 3d - |P_r| - |X|$. (Comment: $Y$ is supposed to be $\bigcup_{i=1}^{4} C_i$. Note that by
    Lemma 5.1, $|P_r| + |P'| - |A_4| + |C_4| + \sum_{i=4}^{4} |C_i| \leqslant 3d$ because $|A_4| > |C_4|$.)
(3) Compute $X_1 = \{s_u \equiv t_r \not\equiv s_r\} \cap X$, $X_2 = \{s_u \equiv s_r \not\equiv t_r\} \cap X$, $X_3 = \{t_r \not\equiv s_r \not\equiv s_u\} \cap X$, $C_1 = X_1 \cap Y$, $C_2 = X_2 \cap Y$,
    $C_3' = X_3 \cap Y$, $A_1 = \{s_u \equiv t_r \not\equiv s_r\} \setminus X_1$, $A_2 = X_1 \setminus C_1$, $A_3 = X_2 \setminus C_2$, $A_4 = \{t_r \not\equiv s_r \not\equiv s_j\} \setminus X_3$, $A_5 = X_3 \setminus C_3'$,
    and $A_6 = P_r \setminus (X \cup A_1 \cup A_4)$. (Comment: $C_3'$ is supposed to be $C_3 \cup C_4$.)
(4) Compute $t_{sol}|_{P_r}$ as follows.
    (4.1) For each position $j \in C_1 \cup C_2 \cup C_3'$, guess $t_{sol}[j] \in \Sigma \setminus \{t_r[j], s_r[j]\}$.
    (4.2) For each $j \in A_2 \cup A_5 \cup A_6$, let $t_{sol}[j] = s_r[j]$.
    (4.3) For each $j \in A_1 \cup A_3 \cup A_4$, let $t_{sol}[j] = t_r[j]$.
```

Fig. 7. Computing $t_{sol}|_{P_r}$ when $|A_4| > |C_4|$.

We are now ready to 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 Fig. 5. 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 $1 \leqslant i \leqslant 3$. 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 $i \in Y \cup C_5$.

We next explain why we need to decide whether $|A_4| \leqslant |C_4|$ or not. By Lemma 5.1, the larger $\sum_{i=1}^{5} |C_i|$ is, the smaller $|B| = d(t_{sol}|_{[1..|t_r|] \setminus (P_r \cup R_u)}, t_r|_{[1..|t_r|] \setminus (P_r \cup R_u)})$ is (and so the shorter time the 2-string algorithm will spend on computing $t_{sol}|_{[1..|t_r|] \setminus (P_r \cup R_u)}$). 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, the idea of 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 $|A_4| \leqslant |C_4|$. Basically, if $|A_4| \leqslant |C_4|$, we *guess* $C_3 \cup A_4$ instead of C_3 only; otherwise, we *guess* $C_3 \cup C_4$ instead of C_3 only. That is, we guess the smaller set between $C_3 \cup A_4$ and $C_3 \cup C_4$.

By the above discussions, there are two ways to compute $t_{sol}|_{P_r}$ (depending on whether $|A_4| \leqslant |C_4|$ or not). They are shown in Figs. 6 and 7, respectively. Their correctness is clear from Fig. 5. To decide which way we should use to compute $t_{sol}|_{P_r}$, we simply guess whether $|A_4| \leqslant |C_4|$ or not. In summary, the details of Step 6 in Fig. 4 are as shown in Fig. 8.

After computing $t_{sol}|_{P_r}$, we construct the string $t' = t_{sol}|_{P_r} + t_r|_{[1..|t_r|] \setminus P_r}$. This construction is similar to Step 7 of the 3-string algorithm for the binary case. If there is no $s \in \mathcal{S}$ with $d(t', s) > d$, then t' is a solution to $(\mathcal{S}, d)$ and we are done. So, suppose that there is an $s \in \mathcal{S}$ with $d(t', s) > d$. We then check whether $d(t', t_r) \leqslant d$, $d(t', s_r) \leqslant d$, and $d(t', s_u) > d$. This checking is similar to that in Step 9 of the 3-string algorithm for the binary case. If this checking fails, we may give up

```
6. Guess whether $|A_4| \leqslant |C_4|$ or not. If the guess was $|A_4| \leqslant |C_4|$, then compute $t_{sol}|_{P_r}$ as in Fig. 6; otherwise,
   compute $t_{sol}|_{P_r}$ as in Fig. 7.
```

Fig. 8. The details of Step 6 in Fig. 4.

```
(1) Guess a subset $R'$ of $R_u$ such that $|R'| \leqslant 3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i|$. (Comment: The upper bound on $|R'|$
    used in this step follows from Lemma 5.1.)
(2) Guess a subset $C_5$ of $R'$ such that $|C_5| \leqslant 3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i| - |R'|$. (Comment: The upper bound
    on $|C_5|$ used in this step follows from Lemma 5.1.)
(3) Compute $A_7 = R_u \setminus R'$ and $A_8 = R' \setminus C_5$.
(4) Compute $t_{sol}|_{R_u}$ as follows.
    (4.1) For each position $j \in C_5$, guess $t_{sol}[j] \in \Sigma \setminus \{t_r[j], s_u[j]\}$.
    (4.2) For each position $j \in A_7$, let $t_{sol}[j] = t_r[j]$.
    (4.3) For each $j \in A_8$, let $t_{sol}[j] = s_u[j]$.
```

Fig. 9. The computation of $t_{sol}|_{R_u}$.

```
10.1. Compute $R_u = \{t_r \equiv s_r \not\equiv s_u\}$.
10.2. If the guess in Step 6 was $|A_4| \leqslant |C_4$, then perform the following steps:
      10.2.1. Compute $A_4 = \{j \in C'_3 \mid t'[j] = t_r[j]\}$, $C_3 = C'_3 \setminus A_4$, and $P' = X$.
      10.2.2. Check whether $|A_4| \leqslant |C_4|$ and $|R_u| \leqslant 3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i|$. If the checking fails, return
              NULL.
10.3. If the guess in Step 6 was $|A_4| > |C_4$, then perform the following steps:
      10.3.1. Compute $C_4 = \{j \in C'_3 \mid t'[j] = s_u[j]\}$, $C_3 = C'_3 \setminus C_4$, and $P' = (X - C_4) \cup A_4$.
      10.3.2. Check whether $|A_4| > |C_4|$ and $|R_u| \leqslant 3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i|$. If the checking fails, return
              NULL.
10.4. Compute $t_{sol}|_{R_u}$ as in Fig. 9.
```

Fig. 10. The details of Step 10 in Fig. 4.

computing $t_{sol}|_{[1..|t_r|]\setminus P_r}$. In fact, we may check more (in order to speed up our algorithm), by distinguishing two cases as follows.

Case 1: *We computed* $t_{sol}|_{P_r}$ *as in Fig.* 6. In this case, we first use t' to compute $A_4 = \{j \in C'_3 \mid t'[j] = t_r[j]\}$ and $C_3 = C'_3 \setminus A_4$. We then check if $|A_4|$ is indeed not larger than $|C_4|$ and $|R_u|$ is indeed not larger than $3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i|$, where $P' = X$.

Case 2: *We computed* $t_{sol}|_{P_r}$ *as in Fig.* 7. In this case, we first use t' to compute $C_4 = \{j \in C'_3 \mid t'[j] = s_u[j]\}$ and $C_3 = C'_3 \setminus C_4$. We then check if $|A_4|$ is indeed larger than $|C_4|$ and $|R_u|$ is indeed not larger than $3d - |P_r| - |P'| - \sum_{i=1}^{4} |C_i|$, where $P' = (X - C_4) \cup A_4$.

In either of the above two cases, if the checking fails, then we know that either t_{sol} does not exist or we did not correctly compute $t_{sol}|_{P_r}$ (because of some incorrect guesses), implying that we may give up computing $t_{sol}|_{[1..|t_r|]\setminus P_r}$. Otherwise, we may proceed to compute $t_{sol}|_{R_u}$ as in Fig. 9. In summary, the details of Step 10 in Fig. 4 are as shown in Fig. 10.

After computing $t_{sol}|_{R_u}$, we construct the string $t = t_{sol}|_{P_r} + t_{sol}|_{R_u} + t_r|_{[1..|t_r|]\setminus(P_r \cup R_u)}$. This construction is similar to Step 9.2 of the 3-string algorithm for the binary case. We then check if $d(t, t_r) \leqslant d$, $d(t, s_r) \leqslant d$, and $d(t, s_u) \leqslant d$. If this checking fails, we know that either the targeted solution t_{sol} does not exist or we did not correctly compute $t_{sol}|_{P_r \cup R_u}$ (because of some incorrect guesses), implying that we may give up computing $t_{sol}|_{[1..|t_r|]\setminus(P_r \cup R_u)}$. Otherwise, we may proceed to compute an upper bound b_2 on the size of $B = \{j \in [1..|t_r|] \setminus (P_r \cup R_u) \mid t_{sol}[j] \neq t[j]\}$.

We can obtain one upper bound on $|B|$ from Lemma 5.1 immediately: $|B| \leqslant (3d - |P_r| - |R_u| - |P'| - |R'| - \sum_{i=1}^{5} |C_i|)/3$. 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 $d(t_r|_{[1..|t_r|]\setminus P_r}, t_{sol}|_{[1..|t_r|]\setminus P_r})$: $b_1 = \min\{d - k_1, k_1 - \ell_r - \sum_{i=1}^{4} |C_i|\}$, where $\ell_r = d(t_r, s_r) - d$ and $k_1 = d(t_r, t')$. 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 $k_2 = |R'|$ and $\ell_2 = d(t_{sol}|_{P_r} + t_r|_{[1..|t_r|]\setminus P_r}, s_u) - d$.

In summary, the details of Step 13 in Fig. 4 are as shown in Fig. 11.

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* $O(nL + dn^2 \cdot (1.612(|\Sigma| + \beta^2 + \beta - 2))^d)$ *time.*

13. Compute $\ell_r = d(t_r, s_r) - d$, $k_1 = d(t_r, t')$, $b_1 = \min\{d - k_1, k_1 - \ell_r - \sum_{i=1}^{4} |C_i|\}$, $k_2 = |R'|$, $\ell_2 = d(t', s_u) - d$, and $b_2 = \min\{b_1 - k_2, k_2 - \ell_2 - |C_5|, (3d - |P_r| - |R_u| - |P'| - |R'| - \sum_{i=1}^{5} |C_i|)/3\}$.

Fig. 11. The details of Step 13 in Fig. 4.

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 7), and finally output t_v in Step 8. 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 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. Fig. 5). $\mathcal{A}_3$ computes $t_v|_{P_r}$ by distinguishing two cases depending on whether $a_4 \leqslant c_4$ or not.

Let us first consider the case where $a_4 \leqslant c_4$. In this case, $\mathcal{A}_3$ performs the steps in Fig. 6. Basically, it guesses P' (as X) in Step (1) and $C_1 \cup C_2 \cup C_3 \cup A_4$ (as Y) in Step (2). In Step (3), it partitions P' into three subsets X_1, X_2, X_3, and further partitions $C_1 \cup C_2 \cup C_3 \cup A_4$ into three subsets C_1, C_2, $C_3 \cup A_4$ (as C_3'). It then uses these subsets to correctly compute $t_v|_{P_r}$ (as $t_{sol}|_{P_r}$) in Step (4).

Next consider the case where $a_4 > c_4$. $\mathcal{A}_3$ performs the steps in Fig. 7. Basically, it guesses $(P' \setminus A_4) \cup C_4$ (as X) in Step (1) and $\bigcup_{i=1}^{4} C_i$ (as Y) in Step (2). In Step (3), it partitions $(P' \setminus A_4) \cup C_4$ into three subsets X_1, X_2, X_3, and further partitions $\bigcup_{i=1}^{4} C_i$ into three subsets C_1, C_2, $C_3 \cup C_4$ (as C_3'). It then uses these subsets to correctly compute $t_v|_{P_r}$ (as $t_{sol}|_{P_r}$) in Step (4).

After computing $t_v|_{P_r}$, $\mathcal{A}_3$ proceed to compute $t_v|_{R_u}$ by performing the steps in Fig. 9. Basically, it guesses R' and C_5 in Steps (1) and (2), respectively. It can then correctly compute $t_v|_{R_u}$ (as $t_{sol}|_{R_u}$) in Step (4).

Once knowing $t_v|_{P_r \cup R_u}$, $\mathcal{A}_3$ can then correctly construct t_v (as t) in Step 11 of Fig. 4, P_v (trivially as $P_r \cup R_u$), and b_v (as b_2) in Steps 13 of Fig. 11. Finally, it calls $\mathcal{A}_2$ to solve $\langle \mathcal{S} \setminus \{t_r, s_r, s_u\}, d; t_v, P_v, b_v\rangle$ in Step 14 of Fig. 4. 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 *type-1 combination* (i.e., a combination of possible outcomes of the guesses made in the details of Step 6 of Fig. 4 such that $\mathcal{A}_3$ does not execute Step 10 of Fig. 4), r has a child corresponding to the combination. Similarly, for each *type*-2 *combination* (i.e., a combination of possible outcomes of the guesses made in the details of Steps 6 and 10 of Fig. 4 such that $\mathcal{A}_3$ executes Step 10), r has a child corresponding to the combination. Moreover, for every child v corresponding to a type-2 combination such that $\mathcal{A}_3$ executes Step 14, 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:

$$N \leqslant 2 \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \tag{5.3}$$

where x, y, i, and j correspond to $|X|$, $|Y|$, $|R'|$, and $|C_5|$ in Figs. 6, 7, and 9, 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 inequality:

$$N' \leqslant 2 \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \binom{d - b_2}{b_2} \alpha^{6b_2}.$$

Hence, by Proposition 4.3,

$$N' \leqslant 2 \cdot 1.612^d \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \alpha^{6b_2}. \tag{5.4}$$

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

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

By Step 14, $|R_u| \leqslant 3d - |P_r| - x - y = h' - x - y$. Moreover, one can verify that $\frac{1}{\alpha^2} + \frac{1}{\alpha^4} + \frac{\gamma}{\alpha^6} \geqslant 1$ and $\beta = 1 + \frac{1}{\alpha^2} + \frac{\gamma}{\alpha^4}$. Thus, by Eq. (5.6), we have

$$\begin{aligned}
N' &\leqslant 2 \cdot 1.612^d \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \left(1 + \frac{1}{\alpha^2} + \frac{\gamma}{\alpha^4}\right)^{h'-x-y}\\
&\leqslant 2 \cdot 1.612^d \cdot \sum_{s_u} \beta^{h'} \sum_{x=0}^{d} \binom{|P_r|}{x} \frac{1}{\beta^x} \sum_{y=0}^{h'-x} \binom{x}{y} \left(\frac{\gamma}{\beta}\right)^y &(5.7)\\
&\leqslant 2 \cdot 1.612^d \cdot \sum_{s_u} \beta^{h'} \sum_{x=0}^{d} \binom{|P_r|}{x} \frac{1}{\beta^x} \left(1 + \frac{\gamma}{\beta}\right)^x\\
&\leqslant 2 \cdot 1.612^d \cdot \sum_{s_u} \beta^{h'} \left(1 + \frac{1}{\beta} + \frac{\gamma}{\beta^2}\right)^{|P_r|}. &(5.8)
\end{aligned}$$

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

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

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 type-1 combination (defined in the 8th paragraph of this proof) satisfies the following inequalities:

$$M_1 \leqslant 2 \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \leqslant 2 \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \beta^{h'-x-y}. \tag{5.10}$$

Note that the right-hand sides of Eqs. (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

$$M_1 \leqslant 2(n-2) \cdot \left(\beta^2 + \beta + \gamma\right)^d. \tag{5.11}$$

On the other hand, the number M_2 of leaves at depth 1 in $\mathcal{T}$ corresponding to a type-2 combination (defined in the 8th paragraph of this proof) satisfies the following inequalities:

$$\begin{aligned}
M_2 &\leqslant 2 \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j\\
&\leqslant 2 \cdot \sum_{s_u} \sum_{x=0}^{d} \binom{|P_r|}{x} \sum_{y=0}^{h'-x} \binom{x}{y} \gamma^y \sum_{i=0}^{f_{x,y}} \binom{|R_u|}{i} \sum_{j=0}^{f_{x,y}-i} \binom{i}{j} \gamma^j \alpha^{6b_2},
\end{aligned}$$

```
9. Check whether $d(t', t_r) \leqslant d$ and $d(t', s_r) \leqslant d$. If the checking fails, return NULL.
```

Fig. 12. The modification of Step 9 in Fig. 4.

```
10.0. If $d(t', s_u) \leqslant d$, 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 $|\{t_r \equiv s_r \not\equiv \tilde{s}_u\}| \leqslant |\{t_r \equiv s_r \not\equiv s\}|$ 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. (Comment: Since $\max\{d(t', t_r), d(t', s_r), d(t', s_u)\} \leqslant d$ but $t'$ is not a solution, $\tilde{s}_u$ must exist.)
```

Fig. 13. A new step to be added immediately before Step 10.1 in Fig. 10.

where we simply let $b_2 = (f_{x,y} - i - j)/3$ to ensure that $\alpha^{6b_2} \geqslant 1$. 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

$$M_2 \leqslant 2(n-2) \cdot \left(\beta^2 + \beta + \gamma\right)^d. \tag{5.12}$$

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

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

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

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

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

As in the binary case (cf. Section 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 Section 4.2. Accordingly, we need to modify Step 9 in Fig. 4 as in Fig. 12. We also need to add the step in Fig. 13 immediately before Step 10.1 of Fig. 10.

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* 10.0 *of Fig.* 13 *but has not let s_u refer to the same string as $\tilde{s}_u$ does. Then, $|P_r| + |P'| + \sum_{i=1}^{4} |C_i| + |\tilde{R}_u| + |\tilde{R}'| \leqslant 3d - 3|\tilde{B}| - |\tilde{C}_5|$, 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, $|\tilde{R}_u| \leqslant |R_u|$. Since $d(t', s_u) = |P'| + |R_u| \leqslant d$ 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, $d(t_{sol}, \hat{s}_u) \leqslant d$ for $d(t_{sol}, \tilde{s}_u) \leqslant d$. Now, applying Lemma 5.1 with s_u there being replaced by $\hat{s}_u$ here, we have $|P_r| + |P'| + |\tilde{R}_u| + |\tilde{R}'| \leqslant 3d - 3|\tilde{B}| - \sum_{i=1}^{4} |C_i| - |\tilde{C}_5|$. Thus, $|P_r| + |P'| + \sum_{i=1}^{4} |C_i| + |\tilde{R}_u| + |\tilde{R}'| \leqslant 3d - 3|\tilde{B}| - |\tilde{C}_5|$. $\square$

Theorem 5.4. *The refined* 3*-string algorithm solves CSP in $O(nL + dn \cdot (1.612(|\Sigma| + \beta^2 + \beta - 2))^d)$ time.*

Proof. To see the correctness of the refined algorithm, it suffices to consider the case where $\tilde{s}_u$ is selected in Step 10.0 of Fig. 13. By Lemma 5.3, the algorithm can correctly guess $\tilde{R}'$ and $\tilde{C}_5$ in Steps (1) and (2) of Fig. 9. The correctness of the computation of the upper bound b_2 on the remaining distance between t and t_{sol} in Step 13 of Fig. 11 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 $O(nL + dn \cdot (1.612(|\Sigma| + \beta^2 + \beta - 2)^d))$ time. $\square$

When $|\Sigma| = 4$, the time bound of the 3-string algorithm given in Theorem 5.2 is $O(nL + nd \cdot 13.183^d)$ which is better than the previously best time bound $O(nL + nd \cdot 13.922^d)$ in [2]. However, when $|\Sigma| \geqslant 7$, 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

The authors thank the anonymous referees for helpful comments. 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 Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 121207).

References

[1] A. Ben-Dor, G. Lancia, J. Perone, R. Ravi, Banishing bias from consensus sequences, in: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching, 1997, pp. 247–261.

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

[3] J. Davila, S. Balla, S. Rajasekaran, Space and time efficient algorithms for planted motif search, in: Proceedings of the International Conference on Computational Science (2), 2006, pp. 822–829.

[4] X. Deng, G. Li, Z. Li, B. Ma, L. Wang, Genetic design of drugs without side-effects, SIAM J. Comput. 32 (4) (2003) 1073–1090.

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

[6] R.G. Downey, M.R. Fellows, Parameterized Complexity, Monogr. Comput. Sci., Springer-Verlag, New York, 1999.

[7] P.A. Evans, A.D. Smith, Complexity of approximating closest substring problems, in: Proceedings of the 14th International Symposium on Foundations of Complexity Theory, 2003, pp. 210–221.

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

[9] M. Frances, A. Litman, On covering problems of codes, Theoret. Comput. Sci. 30 (1997) 113–119.

[10] J. Gramm, J. Guo, R. Niedermeier, On exact and approximation algorithms for distinguishing substring selection, in: Proceedings of the 14th International Symposium on Foundations of Complexity Theory, 2003, pp. 159–209.

[11] J. Gramm, F. Hüffner, 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, 2002, pp. 74–75.

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

[13] Y. Jiao, J. Xu, M. Li, On the k-closest substring and k-consensus pattern problems, in: Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching, 2004, pp. 130–144.

[14] K. Lanctot, M. Li, B. Ma, S. Wang, L. Zhang, Distinguishing string search problems, Inform. and Comput. 185 (2003) 41–55.

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

[16] K. Lucas, M. Busch, S. Mössinger, 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.

[17] B. Ma, X. Sun, More efficient algorithms for closest string and substring problems, SIAM J. Comput. 39 (4) (2010) 1432–1443.

[18] D. Marx, Closest substring problems with small distances, SIAM J. Comput. 38 (4) (2008) 1382–1410.

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

[20] C.N. Meneses, Z. Lu, C.A.S. Oliveira, P.M. Pardalos, Optimal solutions for the closest-string problem via integer programming, INFORMS J. Comput. (2004).

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

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

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

[24] L. Wang, L. Dong, Randomized algorithms for motif detection, J. Bioinform. Comput. Biol. 3 (5) (2005) 1039–1052.

[25] L. Wang, B. Zhu, Efficient algorithms for the closest string and distinguishing string selection problems, in: Proceedings of the 3rd International Frontiers of Algorithmics Workshop, 2009, pp. 261–270.

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

[27] R. Zhao, N. Zhang, A more efficient closest string algorithm, in: Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology, 2010, in press.