Algorithmica (2016) 74:466–484
DOI 10.1007/s00453-014-9952-y

Randomized Fixed-Parameter Algorithms for the Closest String Problem

Zhi-Zhong Chen · Bin Ma · Lusheng Wang

Received: 3 May 2014 / Accepted: 16 October 2014 / Published online: 28 October 2014

Abstract Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of strings of equal length L and an integer d, the *closest string problem* (CSP) requires the computation of a string s of length L such that $d(s, s_i) \leq d$ for each $s_i \in S$, where $d(s, s_i)$ is the Hamming distance between s and s_i. The problem is NP-hard and has been extensively studied in the context of approximation algorithms and fixed-parameter algorithms. Fixed-parameter algorithms provide the most practical solutions to its real-life applications in bioinformatics. In this paper we develop the first randomized fixed-parameter algorithms for CSP. Not only are the randomized algorithms much simpler than their deterministic counterparts, their time complexities are also significantly better than the previously best known (deterministic) algorithms.

A preliminary version of this paper appeared in the *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching*, 2014.

Z.-Z. Chen (✉)
Division of Information System Design, Tokyo Denki University, Ishizaka, Hatoyama, Hiki, Saitama 350-0394, Japan
e-mail: zzchen@mail.dendai.ac.jp

B. Ma
School of Computer Science, University of Waterloo, 200 University Ave. W, Waterloo, ON N2L3G1, Canada
e-mail: binma@uwaterloo.ca

L. Wang
Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR
e-mail: cswangl@cityu.edu.hk

Keywords The closest string problem · Fixed-parameter algorithms · Randomized algorithms · Computational biology

1 Introduction

Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of strings of equal length L and an integer d (called *radius*), the *closest string problem* (CSP) requires the computation of a string s of length L such that $d(s, s_i) \leq d$ for each $s_i \in S$, where $d(s, s_i)$ is the Hamming distance between s and s_i with radius d.

Closest string problem has attracted great attention in recent years due to its important applications in bioinformatics [21]. For example, one needs to solve numerous CSP instances over a binary alphabet in order to find the approximate gene clusters using the Center Gene Cluster model [1,18]. Degenerated Primer Design [34] also involves to solve CSP instances over the DNA alphabet. Other applications include universal PCR primer design [10,16,20,22,30,34], genetic probe design [20], antisense drug design [9,20], finding unbiased consensus of a protein family [3], and gene regulatory motif finding [8,12,16,20,32], etc. Consequently, CSP has been extensively studied in computational biology [9,11,15–17,19–21,24,27–29,31,32]. In particular, CSP has been proved to be NP-hard [14,20].

One approach to CSP is to design approximation algorithms. Along this line, Lancto et al. [20] presented the first non-trivial approximation algorithm for CSP, which achieves a ratio of $\frac{4}{3}$. Li et al. [21] designed the first polynomial-time approximation scheme (PTAS) for CSP. Subsequently, the time complexity of the PTAS was improved in [23,24]. However, the best-known PTAS in [23] has time complexity $\mathcal{O}(Ln^{\mathcal{O}(\epsilon^{-2})})$ which is prohibitive for even a moderately small $\epsilon > 0$.

A more practical approach to CSP is via fixed-parameter algorithms. Fixed-parameter algorithms for CSP are based on the observation that the radius d in a practical instance of CSP is usually reasonably small and hence an algorithm with time complexity $\mathcal{O}(f(d) \times poly(n))$ for a polynomial function $poly(n)$ and exponential function $f(d)$ may still be acceptable. Along this line, Stojanovic et al. [31] designed a linear-time algorithm for the special case of CSP where d is fixed to 1. Gramm et al. [17] proposed the first fixed-parameter algorithm for CSP, which runs in $\mathcal{O}(nL + nd \cdot (d+1)^d)$ time. Ma and Sun [23] designed an algorithm that runs in $\mathcal{O}(nL + nd \cdot (16(|\Sigma| - 1))^d)$ time. This algorithm is the first polynomial-time algorithm for the special case of CSP where d is logarithmic in the input size and the alphabet size $|\Sigma|$ is a constant. Improved algorithms for CSP along this line were given in [6,7,33,35]. Among them, the algorithm with the best theoretical time complexity for general alphabets is given in [7]. For small alphabets, the best time complexity is achieved by the algorithm in [6]. In particular, this algorithm runs in $\mathcal{O}(nL + nd^3 \cdot 6.731^d)$ time for binary strings, while it runs in $\mathcal{O}(nL + nd \cdot 13.183^d)$ time for DNA strings. Noticeably, in order to achieve better time complexity, these best-performing algorithms combined multiple techniques, which made the algorithms rather complicated.

Randomization has been widely employed to design fixed-parameter algorithms for many NP-hard problems [4,5,13,25,26] However, randomization has not been

used to design fixed-parameter algorithms for CSP, and it was unclear if randomization will be of any benefit at all to solving CSP exactly. The only randomized algorithm that we are aware of is a randomized heuristic algorithm for the binary case of CSP proposed by Boucher and Brown [2]. With large synthetic as well as real-genomic data, they demonstrated the heuristic algorithm could detect motifs efficiently. However, no theoretical bounds on the running time or the success probability were provided.

In this paper, we demonstrate that randomization indeed helps us design much simpler and more efficient fixed-parameter algorithms for CSP. Several randomized algorithms are proposed. The first algorithm is presented in Sect. 3 and is for the binary case of CSP. The algorithm is as simple as the following: It starts with a string t that is initialized to s_1. At each iteration it selects an s_i with $d(t, s_i) > d$ and randomly flips one bit of t where s_i disagrees with t. If a center string is not found within d iterations, the algorithm starts over again. This algorithm for binary case uses very similar heuristic as in [2]. However, the procedure to apply the heuristic, as well as the start and end conditions are changed in order to achieve the theoretical bounds proved in this paper. Through rigorous analysis, we show that for any given binary CSP instance, this surprisingly simple algorithm solves the problem in $\mathcal{O}(nL + n\sqrt{d} \cdot (2e)^d) \approx \mathcal{O}(nL + n\sqrt{d} \cdot 5.437^d)$ time with arbitrarily small constant one-sided error, where e is the base of the natural logarithm. This time bound is significantly better than the bound $\mathcal{O}(nL + nd^3 \cdot 6.731^d)$ achieved by the previously best known (deterministic) algorithm for CSP [6].

The algorithm is then extended in the rest of the paper to solve the general-alphabet case and to provide better time complexity. More specifically, the algorithm is slightly changed to solve the general-alphabet case in Sect. 4: Instead of flipping the bit at a randomly selected position of t, the letter at that position is changed to a letter randomly selected from the alphabet according to a carefully designed probability distribution. As a result, we show that CSP can be solved in $\mathcal{O}(nL + n\sqrt{d} \cdot (e\sigma)^d)$ time with arbitrarily small constant one-sided error, where σ is the size of the given alphabet. For DNA strings where $\sigma = 4$, this new time bound is $\mathcal{O}(nL + n\sqrt{d} \cdot (10.874)^d)$, which is significantly better than $\mathcal{O}(nL + nd \cdot 13.183^d)$ achieved by the best known (deterministic) algorithm [6].

In Sect. 5, the algorithm is further improved by a simple strategy that avoids repeated changes at the same position of the candidate center string t. Also, in each iteration the selection of s_i maximizes $d(t, s_i)$. We show that with these small changes, the time complexity of the algorithm is reduced to $\mathcal{O}(nL + n\sqrt{d} \cdot (2.5\sigma)^d)$. Therefore, for binary and DNA strings, the bounds are $\mathcal{O}(nL + n\sqrt{d} \cdot 5^d)$ and $\mathcal{O}(nL + n\sqrt{d} \cdot 10^d)$, respectively.

Noticing that the time complexity $\mathcal{O}(nL + n\sqrt{d} \cdot (2.5\sigma)^d)$ is not better than the algorithm in [7] for large σ, two different strategies are introduced in Sects. 6 and 7 to specifically deal with large alphabets. The algorithm in Sect. 6 runs in $\mathcal{O}(nL + n\sqrt{d} \cdot (2\sigma + 4)^d)$ time. This provides better time complexity than the previously best algorithm in [7] for large σ. For example, for protein strings ($\sigma = 20$), the new algorithm runs in $\mathcal{O}(nL + n\sqrt{d} \cdot 44^d)$ time while the algorithm in [7] runs in $\mathcal{O}(nL + nd \cdot 47.21^d)$ time. The algorithm in Sect. 7 has even better time complexity for large σ. However, the resulting time bound of our analysis is not a closed-form expression.

Via numerical computation, we show that the algorithm runs in $\mathcal{O}(nL + nd^2 \cdot 9.81^d)$ and $\mathcal{O}(nL + nd^2 \cdot 40.1^d)$ time for DNA and protein strings, respectively.

Table 1 in Sect. 7 compares the time complexities of the algorithms in this paper against the previously best-known algorithms for CSP.

2 Notations

In this paper, a string is over an alphabet with a fixed size $\sigma \geq 2$. 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.

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 strings s_1, s_2 of the same length, $\{s_1 \equiv s_2\}$ (respectively, $\{s_1 \not\equiv s_2\}$) denotes the set of all positions where s_1 and s_2 agree (respectively, differ). Moreover, for three strings s_1, s_2, s_3 of the same length, $\{s_1 \not\equiv s_2 \not\equiv s_3\}$ denotes the position set $\{s_1 \not\equiv s_2\} \cap \{s_1 \not\equiv s_3\} \cap \{s_2 \not\equiv s_3\}$, while $\{s_1 \equiv s_2 \not\equiv s_3\}$ denotes the position set $\{s_1 \equiv s_2\} \cap \{s_2 \not\equiv s_3\}$.

3 Randomized Algorithm for Binary Alphabets

To get familiar with the techniques used in this paper, we start with the binary case of the problem in this section.

Most fixed-parameter algorithms for CSP are based on the bounded search tree method. These algorithms start with a suboptimal solution t, and gradually change t to the center string by altering the letters at certain positions of t. At each step, if $d(t, s_i) > d$ for an input string s_i, then at least one of the positions in $\{t \not\equiv s_i\}$ needs to be changed. Different strategies for choosing the position (or positions) to change lead to very different time complexities. Gramm et al. [17] exhaustively tried every position in a size-$(d + 1)$ subset of $\{t \not\equiv s_i\}$, resulting in the first fixed-parameter algorithm for CSP with time complexity $\mathcal{O}\left(nL + nd \cdot (d + 1)^d\right)$. Ma and Sun [23] instead tried every subset of $\{t \not\equiv s_i\}$ with bounded size and changed the positions in a subset all at once, which led to an $\mathcal{O}\left(nL + nd \cdot (16\sigma - 1)^d\right)$ algorithm with a very nontrivial proof. Boucher and Brown [2] proposed a seemingly simple strategy to choose a position from $\{t \not\equiv s_i\}$ uniformly at random. The resulting heuristic algorithm in their paper deviates from the bounded search tree scheme since it can alter the original t more than d times. Although there was no theoretical proof about the better performance of the heuristic algorithm, it worked efficiently on simulated data. Here, we apply the same strategy under the bounded search tree method in Algorithm 1, and show that such a randomized strategy is not only simpler, but also provides a cleaner proof and improves the time complexity.

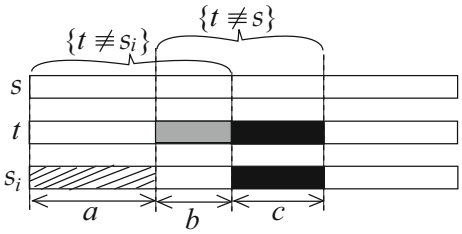

Fig. 1 Strings s, t, and s_i immediately before Step 2.3 of the jth iteration of the while-loop in BoundedGuess-Binary, where for each position $p \in [1 \ldots L]$, two of the strings have the same letter at position p if and only if they are illustrated in the *same color* or pattern at position p

Algorithm 1: BoundedGuess-Binary

Input: Strings $s_1, \ldots, s_n$ each of length L, and a nonnegative integer d.

Output: A center string of $\{s_1, \ldots, s_n\}$ with radius d if there is any, and "no" otherwise.

1 Let $t = s_1$ and $\Delta = d$. If there is a string s_i among the input strings such that $d(t, s_i) > 2d$, then output "no" and halt.

2 While $\Delta > 0$, perform the following steps:

2.1 If $d(t, s_i) \leq d$ for all $1 \leq i \leq n$, output t and halt.

2.2 Find a string s_i among the input strings such that $d(t, s_i) > d$.

2.3 Select a position p in $\{t \not\equiv s_i\}$ uniformly at random and flip the bit of t at position p.

2.4 Decrease Δ by 1.

3 Output "no" and halt.

Lemma 3.1 *If $\{s_1, \ldots, s_n\}$ has no center string of radius d, then BoundedGuess-Binary always outputs "no". On the other hand, if $\{s_1, \ldots, s_n\}$ has a center string of radius d, then with probability at least $\frac{d!}{(2d)^d}$, BoundedGuess-Binary outputs a center string of $\{s_1, \ldots, s_n\}$ with radius d.*

Proof The first assertion in the lemma is obvious. So, suppose that s is a center string of $\{s_1, \ldots, s_n\}$ with radius d such that $d(s, s_1)$ is minimized among all center strings of $\{s_1, \ldots, s_n\}$ with radius d. For convenience, let $d_j = d(s, s_1) - (j - 1)$ for all $j \geq 1$.

For each $j \geq 1$, let E_j be the event that the position p selected in Step 2.3 during the jth iteration of the while-loop belongs to $\{s_1 \not\equiv s\}$, and p has not been selected in Step 2.3 during the previous $j - 1$ iterations of the while-loop. For each $j \geq 1$, we want to obtain a lower bound on the conditional probability $\Pr[E_j \mid E_1, \ldots, E_{j-1}]$.

So, fix an arbitrary $j \geq 1$ and consider the time point immediately before Step 2.3 of the jth iteration of the while-loop. Let $a = |\{t \equiv s \not\equiv s_i\}|$, $b = |\{s_i \equiv s \not\equiv t\}|$, and $c = |\{s_i \equiv t \not\equiv s\}|$ (cf. Fig. 1). Clearly, $d(s_i, s) = a + c \leq d$, $d(t, s) = b + c$,

and $d(t, s_i) = a + b > d$. Let $\ell = d(t, s_i) - d$. Then, $d + \ell + 2c = d(t, s_i) + 2c = a + b + 2c = d(s_i, s) + d(t, s) \leq d + d(t, s)$. If events $E_1, \ldots, E_{j-1}$ have occurred, then $d(t, s) = d_j$ and the last inequality becomes $\ell + 2c \leq d_j$. This implies that $2(d_j - c) \geq d_j + \ell$. Therefore,

$$b = d(t, s) - c = d_j - c \geq \frac{d_j + \ell}{2}. \tag{1}$$

Notice that b is precisely the number of bits in $\{t \not\equiv s_i\}$ where t needs to be flipped in order to reach s. Moreover, if events $E_1, \ldots, E_{j-1}$ have occurred, then none of the b bits has been flipped during the previous $j - 1$ iterations of the while-loop. Thus, $\Pr[E_j \mid E_1, \ldots, E_{j-1}]$ is at least

$$\frac{b}{d + \ell} \geq \frac{1}{2} \cdot \frac{d_j + \ell}{d + \ell} \geq \frac{1}{2} \cdot \frac{d_j}{d} = \frac{d_j}{2d}, \tag{2}$$

where the second inequality is correct because $d_j \leq d$ and $\ell > 0$.

So, the probability that all of $E_1, \ldots, E_{d_1}$ occur is at least

$$\prod_{\Delta=1}^{d_1} \frac{\Delta}{2d} \geq \prod_{\Delta=1}^{d} \frac{\Delta}{2d} = \frac{d!}{(2d)^d},$$

where the inequality holds because $d_1 \leq d$. □

Theorem 3.2 *The binary case of the closest string problem can be solved in* $\mathcal{O}\left(nL + n\sqrt{d} \cdot (2e)^d\right) = \mathcal{O}\left(nL + n\sqrt{d} \cdot 5.437^d\right)$ *time with arbitrarily small constant one-sided error.*

Proof By Stirling's formula, $\frac{d!}{(2d)^d}$ is at least a positive constant times the following:

$$\frac{\sqrt{d} \cdot (\frac{d}{e})^d}{2^d d^d} = \sqrt{d} \left(\frac{1}{2e}\right)^d$$

By Lemma 3.1, if we repeat BoundedGuess-Binary $\mathcal{O}\left(\frac{(2e)^d}{\sqrt{d}}\right) = \mathcal{O}\left(\frac{5.437^d}{\sqrt{d}}\right)$ times, then we will obtain a center string with arbitrarily large constant probability if there is any.

Obviously, each iteration of the while-loop takes $\mathcal{O}(nL)$ time. As observed in previous works (e.g., [17]), we can improve this time bound by carefully remembering the previous distances and only updating them after a single position is flipped. The conclusion is this: With an $\mathcal{O}(nL)$-time preprocessing, each iteration of the while-loop takes $\mathcal{O}(nd)$ time. □

The time bound in Theorem 3.2 is better than $\mathcal{O}\left(nL + nd^3 \cdot 6.731^d\right)$, which is the best time bound achieved by the fastest deterministic algorithm for the binary case [6].

4 Randomized Algorithm for General Alphabets

In this section, we extend the algorithm in Sect. 3 to the general case. In Step 2.3 of Algorithm 1, a randomly selected $p \in \{t \not\equiv s_i\}$ has a good chance to be such that $t[p]$ is different from the center string. Consequently, in the binary case, changing $t[p]$ to $s_i[p]$ will make t one step closer to the center string. However, when the alphabet size is greater than 2, this is not the optimal strategy any more, because $s_i[p]$ can be either equal to or different from the center string. The algorithm has to carefully bet on one of the two cases during the search. Thus, we modify Step 2.3 in Algorithm 1 as follows:

Algorithm 2: BoundedGuess

2.3 Select a position p in $\{t \not\equiv s_i\}$ uniformly at random, change $t[p]$ to $s_i[p]$ with probability $\frac{2}{\sigma}$ while to each of the other $\sigma - 2$ letters with probability $\frac{1}{\sigma}$.

Lemma 4.1 *If $\{s_1, \ldots, s_n\}$ has no center string of radius d, then BoundedGuess always outputs "no". On the other hand, if $\{s_1, \ldots, s_n\}$ has a center string of radius d, then with probability at least $\frac{d!}{(\sigma d)^d}$, BoundedGuess outputs a center string of $\{s_1, \ldots, s_n\}$ with radius d.*

Proof The first assertion in the lemma is obvious. To prove the second assertion, we define s and d_j as in the proof of Lemma 3.1. For each $j \geq 1$, let E_j be the event that the position p selected in Step 2.3 during the jth iteration of the while-loop belongs to $\{s_1 \not\equiv s\}$, p has not been selected in Step 2.3 during the previous $j-1$ iterations of the while-loop, and the letter of t at position p is changed to that of s at position p. The goal is to bound $\Pr[E_j \mid E_1, \ldots, E_{j-1}]$ from below.

So, fix an arbitrary $j \geq 1$ and consider the time point immediately before Step 2.3 of the jth iteration of the while-loop. Let $a = |\{t \equiv s \not\equiv s_i\}|$, $b = |\{s_i \equiv s \not\equiv t\}|$, $c = |\{s_i \equiv t \not\equiv s\}|$, and $r = |\{s_i \not\equiv s \not\equiv t\}|$ (cf. Fig. 2). Clearly, $d(s_i, s) = a + c + r \leq d$, $d(t, s) = b + c + r$, and $d(t, s_i) = a + b + r > d$. Let $\ell = d(t, s_i) - d$. Then, $d + \ell + 2c + r = d(t, s_i) + 2c + r = a + b + 2c + 2r = d(s_i, s) + d(t, s) \leq d + d(t, s)$. If events $E_1, \ldots, E_{j-1}$ have occurred, then $d(t, s) = d_j$ and the above inequality becomes

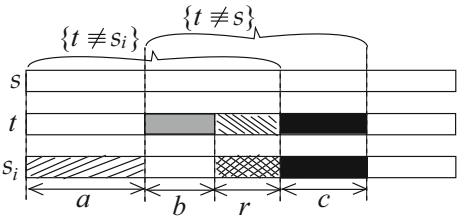

Fig. 2 Strings s, t, and s_i immediately before Step 2.3 of the jth iteration of the while-loop in BoundedGuess, where for each position $p \in [1 \ldots L]$, two of the strings have the same letter at position p if and only if they are illustrated in the *same color* or pattern at position p

$\ell + 2c + r \leq d_j$. Thus,

$$2b + r = 2(d(t, s) - c - r) + r = 2d_j - 2c - r = d_j + (d_j - 2c - r) \geq d_j + \ell. \tag{3}$$

For convenience, let $q_1 = \frac{b}{d+\ell}$ and $q_2 = \frac{r}{d+\ell}$. Then, we have

$$2q_1 + q_2 = \frac{2b + r}{d + \ell} \geq \frac{d_j + \ell}{d + \ell} \geq \frac{d_j}{d}, \tag{4}$$

where the first inequality follows from Eq. (3) and the second inequality is correct because $d_j \leq d$ and $\ell > 0$.

Notice that $b + r$ is precisely the number of positions in $\{t \not\equiv s_i\}$ where t needs to be modified in order to reach s. Among the $b + r$ positions, r need to be changed to one of the $\sigma - 2$ letters that are different from both t and s_i, while b need to be changed to s_i. Moreover, if events $E_1, \ldots, E_{j-1}$ have occurred, then none of the $b + r$ positions has been selected in Step 2.3 during the previous $j - 1$ iterations of the while-loop. Thus, by the total probability rule, $\Pr[E_j \mid E_1, \ldots, E_{j-1}]$ is at least

$$\frac{2}{\sigma} \cdot q_1 + \frac{1}{\sigma} \cdot q_2 = \frac{1}{\sigma} \cdot (2q_1 + q_2) \geq \frac{1}{\sigma} \cdot \frac{d_j}{d}, \tag{5}$$

where the last inequality follows from Eq. (4).

So, the probability that all of $E_1, \ldots, E_{d_1}$ occur is at least

$$\prod_{\Delta=1}^{d_1} \frac{\Delta}{\sigma d} \geq \prod_{\Delta=1}^{d} \frac{\Delta}{\sigma d} = \frac{d!}{(\sigma d)^d},$$

where the inequality holds because $d_1 \leq d$. □

Theorem 4.2 *The closest string problem can be solved in* $\mathcal{O}\left(nL + n\sqrt{d} \cdot (\sigma e)^d\right)$ *time with arbitrarily small constant one-sided error.*

Proof Similar to the proof of Theorem 3.2. The only difference is to use Lemma 4.1 instead of Lemma 3.1. □

For DNA strings ($\sigma = 4$), the time bound $\mathcal{O}\left(nL + n\sqrt{d} \cdot (\sigma e)^d\right) = \mathcal{O}\left(nL + n\sqrt{d} \cdot 10.874^d\right)$ is better than $\mathcal{O}\left(nL + nd \cdot 13.183^d\right)$, which is the best time bound achieved by the fastest deterministic algorithm for the problem [6]. However, for large σ (say, $\sigma = 20$), the time bound in [7] is better.

5 An $\mathcal{O}(nL + n\sqrt{d} \cdot (2.5\sigma)^d)$ Time Algorithm

In this section, we obtain a more efficient algorithm by refining BoundedGuess in Sect. 4.

First, a close inspection of the proof of Lemma 4.1 reveals that we do not need to modify a position of t again once it has been modified. Thus, if we record the already modified positions with a set F, and avoid those positions in later steps, the algorithm's time complexity may be reduced. More specifically, we can augment Step 1 by also initializing $F = \emptyset$, modify Step 2.3 by selecting p from $\{t \not\equiv s_i\} \setminus F$ instead of from $\{t \not\equiv s_i\}$, and augment Step 2.4 by also adding p to F. A crucial observation is that if we find an s_i in Step 2.2 so that $d(t, s_i)$ is maximized, then we can prove that $\{t \not\equiv s_i\} \setminus F$ is significantly smaller than $\{t \not\equiv s_i\}$. Consequently, the probability of selecting a correct position in Step 2.3 is increased. This will be proved in Lemma 5.1.

Secondly, if $d(t, s_i) - d \geq 2$ for the string s_i found in Step 2.2, then we still have $d(t, s_i) > d$ after modifying only one position of t in Step 2.3 and hence the same s_i can be used in Step 2.3 during the next iteration of the while-loop. More generally, if $d(t, s_i) - d = \ell$ for the string s_i found in Step 2.2, then we can use s_i in Step 2.3 during ℓ consecutive iterations of the while-loop.

Based on the above observations, we now obtain a new algorithm as follows:

Algorithm 3: NonRedundantGuess

1 Let $t = s_1$, $\Delta = d$, and $F = \emptyset$. If there is a string s_i among the input strings such that $d(t, s_i) > 2d$, then output "no" and halt.

2 While $\Delta > 0$, perform the following steps:

- **2.1** If $d(t, s_i) \leq d$ for every input string s_i, then output t and halt.
- **2.2** Find a string s_i among the input strings such that $d(t, s_i)$ is maximized.
- **2.3** Compute $\ell = d(t, s_i) - d$.
- **2.4** Select a set R of ℓ positions in $\{t \not\equiv s_i\} \setminus F$ uniformly at random.
- **2.5** For each $p \in R$, change $t[p]$ to $s_i[p]$ with probability $\frac{2}{\sigma}$; and to each of the other $\sigma - 2$ letters with probability $\frac{1}{\sigma}$.
- **2.6** Decrease Δ by ℓ and add the positions in R to F.

3 Output "no" and halt.

To analyze NonRedundantGuess, we first define several notations. For each integer $j \geq 1$, we let s_{i_j} (respectively, ℓ_j or R_j) denote the string s_i (respectively, the integer ℓ or the set R) obtained in Step 2.2 (respectively, Step 2.3 or 2.4) during the jth iteration of the while-loop. Moreover, for each $j \geq 1$, we let t_j (respectively, F_j) denote the string t (respectively, the set F) immediately before the jth iteration of the while-loop.

The next lemma is the key to analyzing NonRedundantGuess.

Lemma 5.1 *For every integer $j \geq 2$, $|F_j \cap \{t_j \not\equiv s_{i_j}\}| \geq \frac{d(t_j, s_1) - \ell_1 + \ell_j}{2}$.*

Proof First, recall that for any two sets A and B,

$$|(A \setminus B) \cup (B \setminus A)| = |A| + |B| - 2|A \cap B|. \tag{6}$$

By the algorithm, $F_j = \{t_1 \not\equiv t_j\}$. So, t_1 and s_{i_j} disagree at each position in $\big(F_j \setminus \{t_j \not\equiv s_{i_j}\}\big) \cup \big(\{t_j \not\equiv s_{i_j}\} \setminus F_j\big)$, and in turn

$$\big|\big(F_j \setminus \{t_j \not\equiv s_{i_j}\}\big) \cup \big(\{t_j \not\equiv s_{i_j}\} \setminus F_j\big)\big| \leq d(t_1, s_{i_j}).$$

Moreover, s_{i_1} maximizes $d(t_1, s_{i_1})$ and so $d(t_1, s_{i_j}) \leq d(t_1, s_{i_1}) = d + \ell_1$. Thus, we have

$$\big|\big(F_j \setminus \{t_j \not\equiv s_{i_j}\}\big) \cup \big(\{t_j \not\equiv s_{i_j}\} \setminus F_j\big)\big| \leq d + \ell_1. \tag{7}$$

Applying Eq. (6) to the left side of Eq. (7), we obtain

$$|F_j| + |\{t_j \not\equiv s_{i_j}\}| - 2|F_j \cap \{t_j \not\equiv s_{i_j}\}| \leq d + \ell_1. \tag{8}$$

Obviously, $|F_j| = d(t_j, s_1)$ and $|\{t_j \not\equiv s_{i_j}\}| = d + \ell_j$. Combining these two equalities with Eq. (8), we finally obtain the inequality in the lemma. □

Lemma 5.2 *If $\{s_1, \ldots, s_n\}$ has no center string of radius d, then NonRedundantGuess always outputs "no". On the other hand, if $\{s_1, \ldots, s_n\}$ has a center string of radius d, then with probability $\Omega(\sqrt{d} \cdot (\frac{(1+\epsilon)^{\frac{1+\epsilon}{2}}(1-\epsilon)^{\frac{1-\epsilon}{2}}}{2^{1+\epsilon}\sigma})^d)$, NonRedundantGuess outputs a center string of $\{s_1, \ldots, s_n\}$ with radius d, where $\epsilon = \ell_1/d$.*

Proof The first assertion in the lemma is obvious. To prove the second assertion, we define s as in the proof of Lemma 3.1. For each $j \geq 1$, let $d_j = d(s, t_j)$. Moreover, for each $j \geq 1$, let E_j be the event that R_j is a subset of $\{s_1 \not\equiv s\}$ and the letter of t at each position $p \in R_j$ is changed to that of s at position p in Step 2.5 during the jth iteration of the while-loop. The goal is to bound $\Pr[E_j \mid E_1, \ldots, E_{j-1}]$ from below.

For convenience, let $R_j = \{p_{j,1}, \ldots, p_{j,\ell_j}\}$ for all $j \geq 1$. Without loss of generality, we can assume that R_j is obtained by the following procedure: For each $j \geq 1$ and $1 \leq h \leq \ell_j$, select $p_{j,h}$ from $\{t_j \not\equiv s_{i_j}\} \setminus \big(F_j \cup \{p_{j,1}, \ldots, p_{j,h-1}\}\big)$ uniformly at random.

For each $j \geq 1$ and $1 \leq h \leq \ell_j$, let $E_{j,h}$ denote the event that $p_{j,h}$ belongs to $\{t_j \not\equiv s\} \setminus \big(F_j \cup \{p_{j,1}, \ldots, p_{j,h-1}\}\big)$ and the letter of t_j at position $p_{j,h}$ is changed to that of s at position $p_{j,h}$ in Step 2.5 during the jth iteration of the while-loop.

By the first inequality in Eq. (4) and the equality in Eq. (5) (cf. the proof of Lemma 4.1), we have $\Pr[E_{1,h} \mid E_{1,1}, \ldots, E_{1,h-1}] \geq \frac{(d_1-(h-1))+(\ell_1-(h-1))}{(d+\ell_1-(h-1))\sigma}$ and in turn

$$\Pr[E_1] \geq \prod_{h=0}^{\ell_1-1} \frac{d_1 + \ell_1 - 2h}{(d + \ell_1 - h)\sigma}. \tag{9}$$

Now, consider an arbitrary integer $j \geq 2$. Since we assume that $E_1, \ldots, E_{j-1}$ occur, $d(t_j, s_1) = d_1 - d_j$. By Lemma 5.1, $|F_j \cap \{t_j \not\equiv s_{i_j}\}| \geq \frac{d_1 - d_j - \ell_1 + \ell_j}{2}$. So, by the first

inequality in Eq. (4), and the equality in Eq. (5) (cf. the proof of Lemma 4.1), we have

$$\Pr[E_{j,h} \mid E_1, \ldots, E_{j-1}, E_{j,1}, \ldots, E_{j,h-1}] \geq \frac{\big(d_j - (h-1)\big) + \big(\ell_j - (h-1)\big)}{\Big(d + \ell_j - \frac{d_1 - d_j - \ell_1 + \ell_j}{2} - (h-1)\Big)\sigma}$$
$$= \frac{2\big(d_j + \ell_j - 2(h-1)\big)}{\big(2d - d_1 + \ell_1 + d_j + \ell_j - 2(h-1)\big)\sigma} \geq \frac{2\big(d_j - (h-1)\big)}{\big(2d - d_1 + \ell_1 + d_j - (h-1)\big)\sigma},$$

where the last inequality is because $\ell_j - (h-1) \geq 0$. In turn,

$$\Pr[E_j \mid E_1, \ldots, E_{j-1}] \geq \prod_{h=0}^{\ell_j - 1} \frac{2\big(d_j - h\big)}{\big(2d - d_1 + \ell_1 + d_j - h\big)\sigma}$$
$$= \prod_{i=d_j - \ell_j + 1}^{d_j} \frac{2i}{(2d - d_1 + \ell_1 + i)\sigma}. \tag{10}$$

Therefore, $\Pr[E_1 \wedge E_2 \wedge \cdots]$ is at least

$$\prod_{i=0}^{\ell_1 - 1} \frac{d_1 + \ell_1 - 2i}{(d + \ell_1 - i)\sigma} \cdot \prod_{i=1}^{d_1 - \ell_1} \frac{2i}{(2d - d_1 + \ell_1 + i)\sigma}$$
$$= \left(\frac{2}{\sigma}\right)^{d_1} \cdot \prod_{i=0}^{\ell_1 - 1} \frac{\frac{d_1 + \ell_1}{2} - i}{(d + \ell_1 - i)} \cdot \prod_{i=1}^{d_1 - \ell_1} \frac{i}{(2d - d_1 + \ell_1 + i)}$$
$$= \left(\frac{2}{\sigma}\right)^{d_1} \cdot \frac{\left(\frac{d_1 + \ell_1}{2}\right)!\, d!\ (d_1 - \ell_1)!\, (2d - d_1 + \ell_1)!}{\left(\frac{d_1 - \ell_1}{2}\right)!\ (d + \ell_1)!\, (2d)!} \tag{11}$$

For convenience, let $\delta = \frac{d_1}{d}$. By the triangle inequality, $d(s_1, s_{i_1}) \leq d(s, s_1) + d(s, s_{i_1}) = d_1 + d(s, s_{i_1}) \leq d_1 + d$ and in turn $\ell_1 = d(s_1, s_{i_1}) - d \leq d_1$. So, $\delta \geq \epsilon$.

Now, by Eq. (11) and Stirling's formula, the probability is at least a positive constant times the following:

$$\sqrt{d} \cdot \left(\frac{2^{\delta - \epsilon}}{4\sigma^{\delta}} \cdot \frac{(\delta + \epsilon)^{\frac{\delta + \epsilon}{2}} (\delta - \epsilon)^{\frac{\delta - \epsilon}{2}} (2 - \delta + \epsilon)^{2 - \delta + \epsilon}}{(1 + \epsilon)^{1 + \epsilon}}\right)^d \tag{12}$$

By elementary calculus, one can verify that Eq. (12) is a decreasing function of δ for $\delta \geq \epsilon$. Therefore Eq. (12) reaches its minimum value when $\delta = 1$. Consequently, the probability is at least a positive constant times the following:

$$\sqrt{d} \cdot \left(\frac{(1 + \epsilon)^{\frac{1 + \epsilon}{2}} (1 - \epsilon)^{\frac{1 - \epsilon}{2}}}{2^{1 + \epsilon}\sigma}\right)^d \tag{13}$$

This completes the proof. □

Theorem 5.3 *The closest string problem can be solved in*

$$\mathcal{O}\left(nL + n\sqrt{d} \cdot \left(\frac{2^{1+\epsilon}\sigma}{(1+\epsilon)^{\frac{1+\epsilon}{2}}(1-\epsilon)^{\frac{1-\epsilon}{2}}}\right)^d\right)$$

time with arbitrarily small constant one-sided error, where $\epsilon = \ell_1/d$.

Proof Similar to the proof of Theorem 3.2. The only difference is to use Lemma 5.2 instead of Lemma 3.1. □

Corollary 5.4 *The closest string problem can be solved in* $\mathcal{O}\left(nL + n\sqrt{d} \cdot (2.5\sigma)^d\right)$ *time with arbitrarily small constant one-sided error.*

Proof By elementary calculus, one can verify that for a fixed σ, Eq. (13) achieves the minimum value $\sqrt{d} \cdot \left(\frac{2}{5\sigma}\right)^d$ at $\epsilon = 0.6$. This completes the proof. □

The time bound in Corollary 5.4 is better than that in Theorem 4.2.

6 More Efficient Algorithm for Large Alphabets

When $\sigma < 16$, the time complexity in Corollary 5.4 is better than the previously best known algorithms [6,7]. However, for large σ (such as $\sigma = 20$ for protein sequences), the algorithm in [7] is still better. In this section, we refine NonRedundantGuess to obtain a new algorithm (named *LargeAlphabet1*) that has a time complexity $\mathcal{O}(nL + n\sqrt{d} \cdot (2\sigma + 4)^d)$. This time complexity is better than the one of Corollary 5.4 for $\sigma \geq 9$. Also, the new time complexity is smaller than that of the algorithm in [7] for reasonably large alphabet sizes (such as $\sigma = 20$).

When the alphabet size is large, the most expensive factor in the time complexity in Corollary 5.4 is the σ^d. This factor arises from the fact that for each position that needs to be changed in Step 2.5 of NonRedundantGuess, we need to guess the letter from $\sigma - 1$ possibilities and in total there can be as many as d such guessing events. However, in the first iteration of the while-loop of NonRedundantGuess, there can be a large number of positions $p \in \{t \not\equiv s_i\}$ such that $t[p]$ needs to be changed to $s_i[p]$. Denote the set of these positions by B. For the positions in B, the algorithm actually does not need to guess the letters. Moreover, by Lemma 3.1 in [7], $|B| \geq \ell_1$. Based on this fact, we obtain LargeAlphabet1 by modifying Step 2.5 of NonRedundantGuess as follows:

Algorithm 4: LargeAlphabet1

2.5 If $\Delta = d$, then change $t[p]$ to $s_i[p]$ for each $p \in R$. Otherwise, for each $p \in R$, change $t[p]$ to $s_i[p]$ with probability $\frac{2}{\sigma}$; and to each of the other $\sigma - 2$ letters with probability $\frac{1}{\sigma}$.

Lemma 6.1 *If $\{s_1, \ldots, s_n\}$ has no center string of radius d, then LargeAlphabet1 always outputs "no". On the other hand, if $\{s_1, \ldots, s_n\}$ has a center string of radius d, then with probability $\Omega\left(d \cdot \sqrt{\epsilon(1-\epsilon)} \cdot \left(\frac{\epsilon^{\epsilon}(1-\epsilon)^{1-\epsilon}}{2^{1+\epsilon}\sigma^{1-\epsilon}}\right)^{d}\right)$, LargeAlphabet1 outputs a center string of $\{s_1, \ldots, s_n\}$ with radius d, where $\epsilon = \ell_1/d$.*

Proof We inherit the notations from Sect. 5. The first assertion in the lemma is obvious. To prove the second assertion, we first observe that $\{s_1 \not\equiv s_{i_1}\}$ contains at least ℓ_1 positions p such that $s_1[p]$ should be changed to $s_{i_1}[p]$. This follows from Lemma 3.1 in [7]. Consequently, we have

$$\Pr[E_1] \geq \frac{1}{\binom{d+\ell_1}{\ell_1}} = \frac{\ell_1!\, d!}{(d+\ell_1)!}. \tag{14}$$

Obviously, for $j \geq 2$, Eq. (10) in the proof of Lemma 5.2 still holds. Therefore, $\Pr[E_1 \wedge E_2 \wedge \cdots]$ is at least

$$\begin{aligned}\frac{\ell_1!\, d!}{(d+\ell_1)!} \cdot \prod_{i=1}^{d_1-\ell_1} \frac{2i}{(2d-d_1+\ell_1+i)\sigma} &\geq \frac{\ell_1!\, d!}{(d+\ell_1)!} \cdot \prod_{i=1}^{d-\ell_1} \frac{2i}{(d+\ell_1+i)\sigma} \\ &= \frac{d!\, \ell_1!\, (d-\ell_1)!}{(2d)!} \cdot \left(\frac{2}{\sigma}\right)^{d-\ell_1}, \end{aligned} \tag{15}$$

where the inequality can be easily verified because the inequality $d_1 \leq d$ implies that $(d_1-\ell_1)! \cdot (2d-d_1+\ell_1)! \geq (d-\ell_1)! \cdot (d+\ell_1)!$. Therefore, by Stirling's formula, $\Pr[E_1 \wedge E_2 \wedge \cdots]$ is $\Omega\left(d \cdot \sqrt{\epsilon(1-\epsilon)} \cdot \left(\frac{\epsilon^{\epsilon}(1-\epsilon)^{1-\epsilon}}{2^{1+\epsilon}\sigma^{1-\epsilon}}\right)^{d}\right)$. □

Theorem 6.2 *The closest string problem can be solved in*

$$\mathcal{O}\left(nL + n \cdot \frac{1}{\sqrt{\epsilon(1-\epsilon)}} \cdot \left(\frac{2^{1+\epsilon}\sigma^{1-\epsilon}}{\epsilon^{\epsilon}(1-\epsilon)^{1-\epsilon}}\right)^{d}\right)$$

time with arbitrarily small constant one-sided error, if $0 < \epsilon = \ell_1/d < 1$.

Proof Similar to the proof of Theorem 3.2. The only difference is to use Lemma 6.1 instead of Lemma 3.1. □

Corollary 6.3 *The closest string problem can be solved in $\mathcal{O}\left(nL + n\sqrt{d} \cdot (2\sigma+4)^{d}\right)$ time with arbitrarily small constant one-sided error.*

Proof If $\epsilon = 0$, then $\ell_1 = 0$ and so LargeAlphabet1 takes $\mathcal{O}(nL)$ time because s_1 itself is a center string of $\{s_1, \ldots, s_n\}$ with radius d. Moreover, if $\epsilon = 1$, then $\ell_1 = d$ and so the equality in Eq. (15) implies that LargeAlphabet1 takes $\mathcal{O}\left(nL + n\sqrt{d} \cdot 4^{d}\right)$ time. So, assume that $0 < \epsilon < 1$. Then, $\frac{1}{\epsilon(1-\epsilon)} = \mathcal{O}(d)$. Consider the function

$f(\epsilon) = \frac{2^{1+\epsilon}\sigma^{1-\epsilon}}{\epsilon^{\epsilon}(1-\epsilon)^{1-\epsilon}}$. By elementary calculus, one can verify that f takes the maximum value $2\sigma + 4$ at $\epsilon = \frac{2}{\sigma+2}$. Now, the corollary follows from Theorem 6.2. □

Obviously, when $\sigma \geq 9$, the time bound in Corollary 6.3 is better than that in Corollary 5.4. In particular, for protein strings (i.e., when $\sigma = 20$), the time bound in Corollary 6.3 becomes $\mathcal{O}\left(nL + n\sqrt{d} \cdot 44^d\right)$, which is better than the best time bound $\mathcal{O}\left(nL + nd \cdot 47.21^d\right)$ achieved by a deterministic algorithm for the problem [7].

Since the value of $\epsilon = \ell_1/d$ in Theorems 5.3 and 6.2 is known at the very beginning of NonRedundantGuess and LargeAlphabet1, one can choose between the two algorithms depending on which of the two bounds in Theorems 5.3 and 6.2 is smaller. For $\sigma \geq 5$, numerical computation shows that this combined algorithm has a better time complexity than the bounds proved in Corollaries 5.4 and 6.3. For example, when $\sigma = 20$, we can show that the combined algorithm has time complexity $\mathcal{O}\left(nL + n\sqrt{d} \cdot 43.54^d\right)$.

7 More Efficient Algorithm for Nonbinary Alphabets

In this section, we refine NonRedundantGuess in a different way to obtain a new algorithm (named *LargeAlphabet2*). The improvement has two consequences. First, LargeAlphabet2 runs faster than the algorithm in [7] for every $\sigma \geq 2$; whereas LargeAlaphabet1 developed in the previous section has a higher complexity than the algorithm in [7] for very large alphabets ($\sigma > 36$). Secondly, we will show that for $\sigma \geq 3$, the combination of NonRedundantGuess and LargeAlphabet2 has a better time bound than all other bounds obtained in this paper and the algorithm in [7]. However, the only drawback of LargeAlphabet2 is that we are unable to obtain a closed-form expression for its time complexity.

We inherit the notations from Sect. 5. Recall that the idea behind LargeAlphabet1 is as follows: In the first iteration of the while-loop, we first randomly guess ℓ_1 positions from $\{t \not\equiv s_i\}$ and then change the letter of t at each guessed position p to $s_i[p]$. The idea behind LargeAlphabet2 is to randomly guess more in the first iteration. More specifically, in the first iteration, we guess $B = \{s_i \equiv s \not\equiv t\}$, $H = \{t \not\equiv s_i \not\equiv s\}$, and $s[p]$ for each $p \in H$. The crucial point is that after we change $t[p]$ to $s[p]$ for each $p \in B \cup H$ in the first iteration, we can decrease Δ down to $\min\{d - |B| - |H|, |B| - \ell_1\}$ (instead of down to $d - |B| - |H|$). This follows from Lemma 3.1 in [7].

Based on the discussion in the last paragraph, we obtain LargeAlphabet2 from NonRedundantGuess by replacing Steps 2.4 through 2.6 with the next step:

Lemma 7.1 *Let ϵ be as in Lemma 5.2. If $\{s_1, \ldots, s_n\}$ has no center string of radius d, then LargeAlphabet2 always outputs "no". On the other hand, if $\{s_1, \ldots, s_n\}$ has a center string of radius d, then with probability at least $\Omega\left(\frac{\gamma^d}{d}\right)$, LargeAlphabet2*

Algorithm 5: LargeAlphabet2

2.4 If $\Delta < d$, then perform Steps 2.4, 2.5, and 2.6 of NonRedundantGuess.
Otherwise, perform the following three steps:

2.4.1 Select an integer $h \in \{0, 1, \ldots, d-\ell\}$, a subset H of $\{t \not\equiv s_i\}$ with $|H| = h$, and a subset B of $\{t \not\equiv s_i\} \setminus H$ with $\ell \le |B| \le d-h$ all uniformly at random.

2.4.2 Let $R = B \cup H$. For each $p \in R$,change $t[p]$ to $s_i[p]$ if $p \in B$, while change $t[p]$ to one of the $\sigma - 2$ letters not equal to $s_i[p]$ uniformly at random if $p \in H$.

2.4.3 Set $\Delta = \min\{d - |R|, |B| - \ell\}$ and $F = R$. [1]

[1] Instead of setting $F = R$, we can set $F = \{t \not\equiv s_i\}$. This change can only speed up the algorithm. The reason for setting $F = R$ is for the clarity of the proof by using Lemma 5.1

outputs a center string of $\{s_1, \ldots, s_n\}$ *with radius* d*, where*

$$\gamma = \begin{cases} \frac{(1-\epsilon)^{\frac{1-\epsilon}{2}}(1+\epsilon)^{1+\epsilon}}{(3+\epsilon)^{\frac{3+\epsilon}{2}}} & \text{if } \sigma = 2 \\ \min\limits_{0\le\alpha\le 1-\epsilon} \frac{1}{(\sigma-2)^{\alpha}} \left(\frac{1-\epsilon-\alpha}{\sigma}\right)^{\frac{1-\epsilon-\alpha}{2}} \frac{2^{\frac{1-\epsilon+\alpha}{2}} \alpha^{\alpha}(1+\epsilon-\alpha)^{(1+\epsilon-\alpha)}}{(3+\epsilon-\alpha)^{\frac{3+\epsilon-\alpha}{2}}} & \text{otherwise.} \end{cases} \tag{16}$$

Proof We inherit the notations from Sect. 5. The first assertion in the lemma is obvious. To prove the second assertion, we first calculate $\Pr[E_0 \wedge E_1]$, where E_0 is the event $R_1 = \{s_1 \not\equiv s\} \cap \{s_1 \not\equiv s_{i_1}\}$. Clearly, both events E_0 and E_1 occur if and only if

1. The set B selected in Step 2.4.1 is $\{s_{i_1} \equiv s \not\equiv s_1\}$,
2. The set H selected in Step 2.4.1 is $\{s_1 \not\equiv s_{i_1} \not\equiv s\}$, and
3. $t[p]$ is changed to $s[p]$ in Step 2.4.2 for each $p \in H$.

For convenience, let $b_1 = |\{s_{i_1} \equiv s \not\equiv s_1\}|$ and $h_1 = |\{s_1 \not\equiv s_{i_1} \not\equiv s\}|$. Note that the number of different combinations for B, H, and the letters in H is bounded from above by

$$d \cdot \binom{d+\ell_1}{h_1} \cdot (\sigma-2)^{h_1} \cdot \sum_{b=\ell_1}^{d-h_1} \binom{d+\ell_1-h_1}{b} \le d \cdot \binom{d+\ell_1}{h_1} (\sigma-2)^{h_1} 2^{d+\ell_1-h_1}.$$

Thus, $\Pr[E_0 \wedge E_1] \ge \frac{1}{d(\sigma-2)^{h_1}2^{d+\ell_1-h_1}} \cdot \frac{h_1!(d+\ell_1-h_1)!}{(d+\ell_1)!}$. So, if $h_1 = 0$, then

$$\Pr[E_0 \wedge E_1] \ge \frac{1}{d \cdot 2^{d+\ell_1}} \tag{17}$$

Otherwise, by Stirling's formula, $\Pr[E_0 \wedge E_1]$ is at least a positive constant times the following:

$$\sqrt{\frac{h_1(d+\ell_1-h_1)}{d+\ell_1}} \cdot \frac{h_1^{h_1}(d+\ell_1-h_1)^{d+\ell_1-h_1}}{d(\sigma-2)^{h_1}2^{d+\ell_1-h_1}(d+\ell_1)^{d+\ell_1}} \tag{18}$$

Let $d_1 = d(s_1, s) \le d$ be the precise number of changes required to convert s_1 to s. Suppose that both E_0 and E_1 have occurred. Then, Lemma 3.1 in [7] guarantees that after the first iteration of the while-loop, we need to further modify at most $\min(d_1 - b_1 - h_1, b_1 - \ell_1) \le \frac{d_1-h_1-\ell_1}{2}$ positions of t. Thus, after the first iteration, Δ is an upper bound on the remaining necessary changes to t, and F consists of the already fixed positions of t that will be excluded from future changes. Hence, after the first iteration, the behavior of LargeAlphabet2 will be the same as that of NonRedundantGuess. Therefore, by Lemma 5.1 and Eq. (10), we know that for each $j \ge 2$,

$$\Pr[E_j \mid E_0, E_1, \ldots, E_{j-1}] \ge \prod_{i=d_j-\ell_j+1}^{d_j} \frac{2i}{(2d - d_1 + \ell_1 + i)\sigma}. \tag{19}$$

Let $m = \lfloor \frac{d_1-h_1-\ell_1}{2} \rfloor$. Then, $\Pr[E_2 \wedge E_3 \wedge \cdots \mid E_0, E_1]$ is bounded from below by

$$\prod_{i=1}^{\lfloor \frac{d_1-h_1-\ell_1}{2} \rfloor} \frac{2i}{(2d - d_1 + \ell_1 + i)\sigma} = \frac{2^m m!(2d - d_1 + \ell_1)!}{\sigma^m (2d - d_1 + \ell_1 + m)!}. \tag{20}$$

Note that if $m = 0$, then the right side of Eq. (20) is 1. So, we may assume that $m \ge 1$ and hence $d_1 - h_1 - \ell_1 \ge 2$. Moreover, by Stirling's formula, Eq. (20) is at least a positive constant times the following:

$$\frac{2^m m^m (2d - d_1 + \ell_1)^{2d-d_1+\ell_1}}{\sigma^m (2d - d_1 + \ell_1 + m)^{2d-d_1+\ell_1+m}} \cdot \frac{\sqrt{2\pi m(2d - d_1 + \ell_1)}}{\sqrt{2d - d_1 + \ell_1 + m}} \tag{21}$$

By elementary calculus, one can easily verify that the first factor in Eq. (21) is a decreasing function in m, and the second factor is at least 1. Thus, Eq. (21) is at least

$$\frac{2^{\tilde{m}} \tilde{m}^{\tilde{m}} (2d - d_1 + \ell_1)^{2d-d_1+\ell_1}}{\sigma^{\tilde{m}} (2d - d_1 + \ell_1 + \tilde{m})^{2d-d_1+\ell_1+\tilde{m}}} \tag{22}$$

where $\tilde{m} = \frac{d_1-h_1-\ell_1}{2}$.

We claim that Eq. (22) is a decreasing function in d_1. To see this claim, let $f(d_1)$ denote the logarithm of Eq. (22). It suffices to show that $f(d_1)$ is a decreasing function in d_1. We have

$$f'(d_1) = \frac{1}{2} \ln \frac{2}{\sigma} + \ln \frac{\sqrt{\tilde{m}(2d - d_1 + \ell_1 + \tilde{m})}}{2d - d_1 + \ell_1} \le \frac{1}{2} \ln \frac{2}{\sigma} + \ln \frac{2d - d_1 + \ell_1 + 2\tilde{m}}{2(2d - d_1 + \ell_1)}.$$

Obviously, $2 \le \sigma$. Moreover, $2d - d_1 + \ell_1 + 2\tilde{m} = 2d - h_1 \le 2(2d - d_1 + \ell_1)$ for $d_1 \le d$. So, $f'(d_1) \le 0$ and in turn $f(d_1)$ is a decreasing function in d_1. This completes the proof of the claim.

By the claim, Eq. (22) is at least

$$\frac{(d-h_1-\ell_1)^{\frac{d-h_1-\ell_1}{2}}2^{\frac{3d+\ell_1-h_1}{2}}(d+\ell_1)^{d+\ell_1}}{\sigma^{\frac{d-h_1-\ell_1}{2}}(3d+\ell_1-h_1)^{\frac{3d+\ell_1-h_1}{2}}}.$$

Hence, by Eq. (17) and 18, $\Pr[E_0 \wedge E_1 \wedge E_2 \wedge \cdots]$ is

$$\Omega\left(\frac{(d-h_1-\ell_1)^{\frac{d-h_1-\ell_1}{2}}h_1^{h_1}(d+\ell_1-h_1)^{d+\ell_1-h_1}2^{\frac{d-\ell_1+h_1}{2}}}{d\cdot\sigma^{\frac{d-h_1-\ell_1}{2}}(\sigma-2)^{h_1}(3d+\ell_1-h_1)^{\frac{3d+\ell_1-h_1}{2}}}\right) \tag{23}$$

Therefore, if we let $\alpha = \frac{h_1}{d}$ and $\epsilon = \frac{\ell_1}{d}$, then

$$\Pr[E_0 \wedge E_1 \wedge E_2 \wedge \cdots] = \Omega\left(\frac{\gamma^d}{d}\right),$$

where γ is as defined in Eq. (16). □

Thus, we have the following theorem.

Theorem 7.2 *The closest string problem can be solved in $\mathcal{O}(nL + nd^2 \cdot (\frac{1}{\gamma})^d)$ time with arbitrarily small constant one-sided error, where γ is as defined in Eq. (16).*

Proof Similar to the proof of Theorem 3.2. The only difference is to use Lemma 7.1 instead of Lemma 3.1. □

Table 1 The time complexities of the previously best-known algorithms and the algorithms developed in this paper.

Algorithm	Reference	$c = f(\sigma)$	$\sigma = 2$	$\sigma = 4$	$\sigma = 20$	$\sigma = 50$
3-String	[6]	See caption	**6.73**	**13.18**	51.23	113.3
CloseString2	[7]	See caption	8	13.92	**47.21**	**100.4**
BoundedGuess	Theorem 4.2	$e\sigma$	5.44	10.87	54.37	135.9
NonRedundantGuess	Corollary 5.4	2.5σ	**5**	10	50	125
LargeAlphabet1	Corollary 6.3	$2\sigma + 4$	8	12	44	104
LargeAlphabet2	Theorem 7.2	–	5.83	10.47	40.13	**86.84**
NonRedundantGuess + LargeAlphabet2	Sect. 7	–	**5**	**9.81**	**40.09**	**86.84**

Here, if the time complexity of an algorithm is of the form $poly(n, L) \cdot c^d$ for some constant c and a polynomial $poly(n, L)$, then the row of the table corresponding to the algorithm shows the values of c at different alphabet sizes. "NonRedundantGuess+LargeAlphabet2" means the algorithm that emulates the two algorithms in parallel and halts once one of them stops. If a closed-form expression exists for an algorithm, the column "$c = f(\sigma)$" shows c as a function of σ. For the 3-string algorithm in [6], $c = 1.612\left(\sigma + \beta^2 + \beta - 2\right)$ in general where $\beta = \alpha^2 + 1 - 2\alpha^{-1} + \alpha^{-2}$ with $\alpha = \sqrt[3]{\sqrt{\sigma - 1} + 1}$, but its time bound for $\sigma = 2$ has a better analysis. For the algorithm in [7], $c = \sqrt{2}\sigma + \sqrt[4]{8}\left(\sqrt{2} + 1\right)\left(1 + \sqrt{\sigma - 1}\right) - 2\sqrt{2}$ in general, but its time bound for $\sigma = 2$ has a better analysis
Bold values indicate either the previously best or the currently best bound

Although it is difficult to find a closed-form expression for the value of $\frac{1}{\gamma}$, we can perform numerical computation to obtain an approximate upper bound on $\frac{1}{\gamma}$ for any given σ. Row "LargeAlphabet2" in Table 1 shows the results for several specific σ. Note that the time bound in Theorem 7.2 is not as good as those in Corollaries 5.4 and 6.3 for small σ (such as $\sigma = 4$) but is significantly better for large σ (say, $\sigma = 20$ or 50). Furthermore, numerical computation also shows that for every $\sigma \geq 2$, the bound in Theorem 7.2 is better than the time bound in [7].

For $\sigma \geq 3$, numerical computation shows that the smaller bound between the two in Theorems 5.3 and 7.2 is better than the bounds in Corollaries 5.4 and 6.3 and Theorem 7.2. For example, when $\sigma = 4$ (respectively, $\sigma = 20$), we can show that the smaller bound between the two in Theorems 5.3 and 7.2 is $\mathcal{O}\left(nL + nd^2 \cdot 9.81^d\right)$ [respectively, $\mathcal{O}\left(nL + nd^2 \cdot 40.1^d\right)$].

Acknowledgments We thank the anonymous referees for very 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. 24500023. Bin Ma was supported in part by Natural Sciences and Engineering Research Council of Canada (RGPIN 238748). Lusheng Wang was supported by a GRF grant from Hong Kong SAR government Project No. [CityU 123013] and a grant from National Foundation of China Project No. [61373048].

References

1. Böcker, S., Jahn, K., Mixtacki, J., Stoye, J.: Computation of median gene clusters. J. Comput. Biol. **16**(8), 1085–1099 (2009)
2. Boucher, C., Brown, D.: Detecting motifs in a large data set: applying probabilistic insights to motif finding. In: Proceedings of the Conference on Bioinformatics and Computational Biology (BICoB), pp. 139–150 (2009)
3. Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus sequences. In: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching, pp. 247–261 (1997)
4. Chen, J., Lu, S.: Improved parameterized set splitting algorithms: a probabilistic approach. Algorithmica **54**(4), 472–489 (2008)
5. Chen, J., Lu, S., Sze, S.H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 298–307 (2007)
6. Chen, Z.-Z., Ma, B., Wang, L.: A three-string approach to the closest string problem. J. Comput. Syst. Sci. **78**, 164–178 (2012)
7. Chen, Z.-Z., Wang, L.: Fast exact algorithms for the closest string and substring problems with application to the planted (ℓ, d)-motif model. IEEE/ACM Trans. Comput. Biol. Bioinform. **8**(5), 1400–1410 (2011)
8. Davila, J., Balla, S., Rajasekaran, S.: Space and time efficient algorithms for planted motif search. In: Proceedings of the International Conference on Computational Science, pp. 822–829 (2006)
9. Deng, X., Li, G., Li, Z., Ma, B., Wang, L.: Genetic design of drugs without side-effects. SIAM J. Comput. **32**(4), 1073–1090 (2003)
10. Dopazo, J., Rodríguez, A., Sáiz, J.C., Sobrino, F.: Design of primers for PCR amplification of highly variable genomes. CABIOS **9**, 123–125 (1993)
11. Evans, P.A., Smith, A.D.: Complexity of approximating closest substring problems. In Proceedings of the 14th International Symposium on Foundations of Complexity Theory, pp. 210–221 (2003)
12. Fellows, M.R., Gramm, J., Niedermeier, R.: On the parameterized intractability of motif search problems. Combinatorica **26**(2), 141–167 (2006)
13. Feng, Q., Wang, J., Li, S., Chen, J.: Random methods for parameterized problems. In: Proceedings of the 19th International Computing and Combinatorics Conference (COCOON), pp. 89–100 (2013)
14. Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Sci. **30**, 113–119 (1997)

15. Gramm, J., Guo, J., Niedermeier, R.: On exact and approximation algorithms for distinguishing substring selection. In: Proceedings of the 14th International Symposium on Foundations of Complexity Theory, pp. 159–209 (2003)
16. Gramm, J., Hüffner, F., Niedermeier, R.: Closest strings, primer design, and motif search. In: Florea, L. et al. (eds.) Currents in Computational Molecular Biology. Poster Abstracts of RECOMB 2002, pp. 74–75
17. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica **37**, 25–42 (2003)
18. Hufsky, F., Kuchenbecker, L., Jahn, K., Stoye, J., Böcker, S.: Swiftly computing center strings. In: Proceedings of the 10th International Workshop on Algorithms in Bioinformatics, pp. 325–336 (2010)
19. Jiao, Y., Xu, J., Li, M.: On the k-closest substring and k-consensus pattern problems. In: Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching, pp. 130–144 (2004)
20. Lanctot, K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string search problems. Inf. Comput. **185**, 41–55 (2003)
21. Li, M., Ma, B., Wang, L.: On the closest string and substring problems. J. ACM **49**(2), 157–171 (2002)
22. Lucas, K., Busch, M., Mösinger, S., Thompson, J.A.: 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)
23. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. SIAM J. Comput. **39**(4), 1432–1443 (2010)
24. Marx, D.: Closest substring problems with small distances. SIAM J. Comput. **38**(4), 1382–1410 (2008)
25. Marx, D.: Randomized techniques for parameterized algorithms. In: Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC), p. 2 (2012)
26. Marx, D., Razgon, I.: Fixed-parameter tractability of multicut parameterized by the size of the cutset. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC), pp. 469–478 (2011)
27. Mauch, H., Melzer, M.J., Hu, J.S.: Genetic algorithm approach for the closest string problem. In: Proceedings of the 2nd IEEE Computer Society Bioinformatics Conference (CSB), pp. 560–561 (2003)
28. Meneses, C.N., Lu, Z., Oliveira, C.A.S., Pardalos, P.M.: Optimal solutions for the closest-string problem via integer programming. INFORMS J. Comput. **16**, 419–429 (2004)
29. Nicolas, F., Rivals, E.: Complexities of the centre and median string problems. In: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching, pp. 315–327 (2003)
30. Proutski, V., Holme, E.C.: Primer master: a new program for the design and analysis of PCR primers. CABIOS **12**, 253–255 (1996)
31. Stojanovic, N., Berman, P., Gumucio, D., Hardison, R., Miller, W.: A linear-time algorithm for the 1-mismatch problem. In: Proceedings of the 5th International Workshop on Algorithms and Data Structures, pp. 126–135 (1997)
32. Wang, L., Dong, L.: Randomized algorithms for motif detection. J. Bioinform. Comput. Biol. **3**(5), 1039–1052 (2005)
33. Wang, L., Zhu, B.: Efficient algorithms for the closest string and distinguishing string selection problems. In: Proceedings of the 3rd International Frontiers of Algorithmics Workshop, pp. 261–270 (2009)
34. Wang, Y., Chen, W., Li, X., Cheng, B.: Degenerated primer design to amplify the heavy chain variable region from immunoglobulin cDNA. BMC Bioinform. **7**(Suppl. 4), S9 (2006)
35. Zhao, R., Zhang, N.: A more efficient closest string algorithm. In: Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology (2010)