Randomized and Parameterized Algorithms for the Closest String Problem

Zhi-Zhong Chen[1], Bin Ma[2], and Lusheng Wang[3]

[1] Division of Information System Design, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan
zzchen@mail.dendai.ac.jp

[2] School of Computer Science, University of Waterloo, 200 University Ave. W, Waterloo, ON, Canada N2L3G1
binma@uwaterloo.ca

[3] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR
cswangl@cityu.edu.hk

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 parameterized algorithms. Parameterized algorithms provide the most practical solutions to its real-life applications in bioinformatics. In this paper we develop the first randomized parameterized algorithms for CSP. Not only are the randomized algorithms much simpler than their deterministic counterparts, their expected-time complexities are also significantly better than the previously best known (deterministic) algorithms.

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. Such a string s is referred to as a *center string* of S with radius d.

CSP has attracted great attention in recently years due to its important applications in bioinformatics [18]. 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 [1,15]. Degenerated Primer Design [30] also involves to solve CSP instances over the DNA alphabet. Other applications include universal PCR primer design [19,17,8,26,13,30], genetic probe design [17], antisense drug design [17,7], finding unbiased consensus of a protein family [3], and gene regulatory motif finding [17,13,28,6,10], etc. Consequently, CSP has been extensively studied in computational biology [17,18,21,14,24,13,23,16,9,12,27,7,25,28]. In particular, CSP has been proved to be NP-hard [11,17].

A.S. Kulikov, S.O. Kuznetsov, and P. Pevzner (Eds.): CPM 2014, LNCS 8486, pp. 100–109, 2014.

One approach to CSP is to design approximation algorithms. Along this line, Lancto *et al.* [17] presented the first non-trivial approximation algorithm for CSP, which achieves a ratio of $\frac{4}{3}$. Li *et al.* [18] designed the first polynomial-time approximation scheme (PTAS) for CSP. Subsequently, the time complexity of the PTAS was improved in [21,20]. However, the best-known PTAS in [20] has time complexity $\mathcal{O}(mn^{\mathcal{O}(\epsilon^{-2})})$ which is prohibitive for even a moderately small $\epsilon > 0$.

A more practical approach to CSP is via parameterized algorithms. Parameterized 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.* [27] designed a linear-time algorithm for the special case of CSP where d is fixed to 1. Gramm *et al.* [14] proposed the first parameterized algorithm for CSP, which runs in $\mathcal{O}(nL + nd \cdot (d+1)^d)$ time. Ma and Sun [20] 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 [29,5,31,4]. Among them, the algorithm with the best theoretical time complexity for general alphabets is given in [5]. For small alphabets, the best time complexity is achieved by the algorithm in [4]. In particular, this algorithm runs in $\mathcal{O}(nL + nd^3 \cdot 6.731^d)$ time for binary strings, while 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 parameterized algorithms for many NP-hard problems [22]. However, randomization has not been used to design parameterized algorithms for CSP, and it is 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 design much simpler and more efficient parameterized algorithms for CSP. Several randomized algorithms are proposed. The first algorithm is presented in Section 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