Iterative Rounding for the Closest String Problem

Jingchao Chen
School of Informatics, Donghua University
2999 North Renmin Road, Songjiang District, Shanghai 201620, P. R. China
chen-jc@dhu.edu.cn

Abstract. The closest string problem is an NP-hard problem, whose task is to find a string that minimizes maximum Hamming distance to a given set of strings. This can be reduced to an integer program (IP). However, to date, there exists no known polynomial-time algorithm for IP. In 2004, Meneses et al. introduced a branch-and-bound (B&B) method for solving the IP problem. Their algorithm is not always efficient and has the exponential time complexity. In this paper, we attempt to solve efficiently the IP problem by a greedy iterative rounding technique. The proposed algorithm is polynomial time and much faster than the existing B&B IP for the CSP. If the number of strings is limited to 3, the algorithm is provably at most 1 away from the optimum. The empirical results show that in many cases we can find an exact solution. Even though we fail to find an exact solution, the solution found is very close to exact solution.

Key words: closest string problem; mathematical programming; NP-problem; integer programming; iterative rounding.

1 Introduction

The task of finding a string that is close to each string in a given set of strings is one of combinatorial optimization problems, which arise in computational molecular biology and coding theory. This problem is called the closest string problem (CSP). We introduce some notations to defining more precisely the CSP. Let \( \Sigma \) stand for a fixed finite alphabet. Its element is called character, and a sequence of characters over it is called string, denoted by \( s \). The length and \( i \)-th character of \( s \) are denoted by \( |s| \) and \( s[i] \), respectively. \( d(s, t) \) is defined as the Hamming distance between two equal-length strings \( s \) and \( t \), i.e. the number of characters where they do not agree. This may be formulated as \( d(s, t) = \sum f(s[i], t[i]) \), where \( f(s[i], t[i]) \) is one if \( s[i] \neq t[i] \), and zero otherwise. Let \( \Sigma^n \) be the set of all strings of length \( n \) over \( \Sigma \). Then, the CSP is defined exactly as follows.

Given a finite set \( S = \{s_1, s_2, \ldots, s_m\} \) of \( m \) strings, each of which is in \( \Sigma^n \), the objective is to find a center string \( t \) of length \( n \) over \( \Sigma \) minimizing the distance \( d \) such that, for every string \( s_i (1 \leq i \leq m) \) in \( S \), \( d(t, s_i) \leq d \).
The CSP has received the attention of many researchers in the recent few years. The literature abounds with the CSP. In theory, Frances and Litman [FL1] have proven that it is NP-hard. However, if the distance \(d\) is fixed, the exact solution to the problem can be found in polynomial time [GN1,BG1]. For the general case where \(d\) is variable, one is involved in studying approximation algorithms. There have been some approximation algorithms with good theoretical precision. For example, Gasieniec et al. [GJ1] and Lanctot et al. [LL1] developed independently a 4/3-approximation algorithm. On the basis of this, Li et al. [LM1] presented a polynomial-time approximation scheme (PTAS). However, the PTAS is not practical.

Meneses et al. [ML1] studied many approximation algorithms, and found that the mentioned-above algorithms are of only theoretical importance, not directly applicable to bioinformatics practice because of high time complexity. For this reason, they suggested reducing the CSP to an integer-programming (IP) problem, and then using branch-and-bound (B&B) algorithm to solve the IP problem. Unfortunately, integer programs are also NP-hard. So far, no polynomial-time algorithm for solving integer programs has been found. Furthermore, the B&B has its own drawbacks. It leads easily to memory explosion due to excessive accumulation of active nodes. In fact, our empirical results show that the B&B IP is not efficient. In despite of instances of moderate size, the B&B IP fails to find an optimal solution sometimes.

We want to find efficiently an exact solution via a technique called iterative rounding. The reason for using this technique is because we noted that Jain [JA1], Cheriyan and Vempala [CV1] used it and succeeded in getting a better approximation algorithm for the generalized steiner network problem. Although our problem is different from their problem, both are NP-hard. Therefore, we believe this technique is applicable to the CSP. The iterative rounding method used here is a greedy one. It may be outlined as follows. First we formulate the CSP as an IP, and then use the LP solution to round some of higher valued variables, finally repeatedly re-solve the LP for the remaining variables until all variables are set. The method has small memory requirement, and can avoid memory explosion of the B&B IP. It is a polynomial time algorithm which can find an exact solution in a very short time for a CSP instance of moderate size in many cases. The computational experiments reveal that our algorithm is not only much faster than the existing one, but also has high quality. If the number of strings is limited to 3, the error of the algorithm is proven to be at most one. Unlike the existing rounding schemes, our rounding scheme is iterative, not random, while the existing ones such as the rounding scheme of Lanctot et al. [LL1] are random. An important contribution of our algorithm is in setting up a new approach for finding the exact CSP algorithm with the polynomial-time.

2 Iterative rounding for the CSP

The CSP can be reduced to a 0-1 Integer Programming problem as follows.
Iterative Rounding for the Closest String Problem

\[ \min d \]
\[ \text{s.t.} \quad \sum_{a \in \Sigma} x_{a,j} = 1 \quad j = 1, \ldots, n \]
\[ n - \sum_{j=1}^{n} x_{a[i,j],j} \leq d \quad i = 1, \ldots, m \]

where \( x_{a,j} \in \{0,1\} \), \( a[i,j] \in \Sigma \), and \( d \) is a non-negative integer. Solving this IP problem by applying directly LP (Linear Programming) relaxation and randomized rounding does not work well because randomized rounding procedure leads to large errors, especially when the optimal distance \( d \) is small [LM1]. Therefore, we decided to find other rounding techniques. Jain [JA1] used iterative rounding to get a 2-approximation algorithm for the generalized steiner network problem. Based on our observation, iterative rounding is suited also for the CSP. Hence, we use it to solve the CSP. The following pseudo-code is a CSP algorithm with iterative rounding.

**Algorithm A**

Formulate the CSP as an IP.

\[ V_1 \leftarrow \text{empty} , \quad V_0 \leftarrow \text{empty} \]

for \( i = 1 \) to \( n \) do

Fix all variables in \( V_1 \) to 1, and all variables in \( V_0 \) to 0

Solve the LP for the sub-CSP on the unfixed variables

Pick a variable \( x_{b,m} \) with highest value, i.e.,

\[ x_{b,m} = \max \{x_{a,k} | k = 1, \ldots, n, a \in \Sigma \text{ and } x_{a,k} \notin V_1\} \]

\( V_1 \leftarrow V_1 \cup \{x_{b,m}\} \)

\( V_0 \leftarrow V_0 \cup \{x_{a,m} | a \neq b \text{ and } a \in \Sigma\} \)

end for

Convert \( V_1 \) into a solution (a center string \( t \)) to the CSP as follows.

\( t[k] \leftarrow a \) for all \( x_{a,k} \in V_1 \).

Clearly, Algorithm A is a polynomial-time algorithm. Furthermore, we have

**Theorem 1.** If the input consists of only two strings, i.e., \( S = \{s_1, s_2\} \), then Algorithm A always find an exact solution to the CSP.

**Proof.** Without loss of generality, we assume

\[ s_1 = \underbrace{000 \ldots 000}_n \]
\[ s_2 = 111 \ldots 111 \]

(Notice, in the case when the same positions of two strings \( s_1, s_2 \) have the same characters, the proof is simpler than in the above case.)

It is easy to see that the 1st LP optimal solution to the CSP is

\[ x_{11}^1 + x_{12}^1 + x_{13}^1 + \cdots + x_{1n}^1 = \frac{n}{2} \]

where \( x_{1k}^1 + x_{1k}^0 = 1 \) for \( k = 1, \ldots, n \).

Without loss of generality, assume \( x_{11}^1 = \max \{x_{a,k}^1 | k = 1, \ldots, n, a \in \{0,1\}\} \).

If \( n \geq 2 \), there exists \( 0 \geq x_{12}, x_{13}, \ldots, x_{1n} \geq 1 \) such that

\[ x_{12} + x_{13} + \cdots + x_{1n} = \frac{n}{2} - 1 \]

Say, \( x_{12} = x_{13} = \cdots = x_{1n} = (n-2)/(2(n-1)) \) is just a solution to this equation.

Then, when \( n \geq 2 \), setting \( x_{11} \) to 1, we can get the 2nd LP optimal solution
By (2), we have that one of the following three propositions is true.

In general, any three strings can be simplified into

Proof. It follows that

Theorem 2. If the input consists of only three binary strings, i.e., \( S = \{s_1, s_2, s_3\} \), then the error of Algorithm A is at most one.

Proof. In general, any three strings can be simplified into

where \( t_0^\alpha \) is the number of 0’s in the \( \alpha \) substring of \( t \), Similarly for \( t_1^\alpha, t_0^\beta, t_1^\beta, t_0^\gamma, t_1^\gamma \).

Assume the distances between \( t \) and the three strings are \( D_1, D_2 \) and \( D_3 \), respectively, we have

The optimal distance is denote by \( D \). Then \( D = \max(D_1, D_2, D_3) \).

the following proposition is true.

If it is false, by (1) we have

It follows that \( D = \max(D_1 - 1, D_2 - 1, D_3 - 1) = D - 1 \), which is a contradiction.

By (2), we have that one of the following three propositions is true.

(a) \( t_0^\beta = 0 \) can constitute a optimal solution, but \( t_0^\beta \neq 0 \) cannot.

(b) \( t_0^\gamma = 0 \) can constitute a optimal solution, but \( t_0^\gamma \neq 0 \) cannot.
(c) $t_0^0 = 0$ can constitute a optimal solution, but $t_0^0 \neq 0$ cannot.

Here we consider only the 2nd case to prove the theorem, since other cases is similar. That is, assume

for any optimal solution, \( t_0^\beta = 0 \)

This implies

\[
D_1 \leq \text{max}(D_2, D_3)
\]

(4)

It is false, (1) can be rewritten as

\[
t_1^\alpha + (t_1^\beta - 1) + t_0^\gamma = D_1 - 1
\]

\[
t_1^\alpha + (t_1^\beta + 1) + t_0^\gamma = D_2 + 1
\]

\[
t_0^\delta + (t_0^\beta + 1) + t_0^\gamma = D_3 + 1
\]

$t_0^\beta + 1$ is also a optimal solution, which is in contradiction with (3).

Without loss of generality, suppose

\[
\alpha \leq \gamma
\]

(5)

If $\alpha > \gamma$, the subsequent proof is similar). This implies

\[
t_1^\alpha \leq t_0^\beta
\]

(6)

If it is false, let $T_1^\alpha = t_1^\alpha - t_0^\gamma$ and $T_0^\alpha = t_0^\beta + t_0^\gamma$, we can rewrite (1) as

\[
T_1^\alpha + t_1^\beta = D_1 - 2t_0^\gamma
\]

\[
T_0^\alpha + t_0^\beta + \gamma = D_2
\]

\[
T_0^\alpha + t_0^\beta = D_3
\]

By (4), we have

\[
D = \text{max}(D_2, D_3) = \text{max}(T_1^\alpha + t_0^\beta + \gamma, T_0^\alpha + t_0^\beta) > \gamma
\]

However, in fact, by fixing $t_1^\alpha = t_0^\beta = t_0^\gamma = 0$, solving (1) yields \((D_1, D_2, D_3) = (\beta, \gamma, \alpha)\). Then by (4) and (5), $D \leq \text{max}(\beta, \gamma, \alpha) \leq \gamma$, which is a contradiction.

By (3) and (6), (1) can be rewritten as

\[
\beta + T_0^\alpha = D_1
\]

\[
T_1^\alpha = D_2
\]

\[
\alpha + T_0^\alpha = D_3
\]

where $T_0^\alpha = t_0^\alpha - t_1^\alpha, T_1^\alpha = t_1^\alpha + t_0^\alpha$ and $D_1^1 = D_1 - 2t_1^\alpha$.

This implies

\[
|D_1 - D_2| \leq 1
\]

(8)

If it is false, by (7), we can obtain a solution with $t_0^\beta = 1$, which is in contradiction with (3).

Let $x_1^1, x_2^2, \ldots, x_n^0$ be 0-variables of the LP, $x_1^1, x_2^1, \ldots, x_n^1$ 1-variables. Define

\[
L_0^\alpha = x_0^1 + x_2^2 + \cdots + x_n^0
\]

\[
L_1^\alpha = x_1^1 + x_2^1 + \cdots + x_n^1
\]

\[
L_0^\beta = x_0^1 + x_2^2 + \cdots + x_n^0
\]

\[
L_1^\beta = x_1^1 + x_2^1 + \cdots + x_n^0
\]

\[
L_0^\gamma = x_0^1 + x_2^2 + \cdots + x_n^0
\]

\[
L_1^\gamma = x_1^1 + x_2^1 + \cdots + x_n^1
\]

Let $d_1, d_2$ and $d_3$ denote the distances between the three strings and the center string of the LP, respectively. Then,

\[
L_1^\alpha + L_1^\beta + L_0^\gamma = d_1
\]

\[
L_1^\alpha + L_0^\beta + L_1^\gamma = d_2
\]

\[
L_0^\alpha + L_0^\beta + L_0^\gamma = d_3
\]

(9)
and \( \gamma \) be the number of letters in the substring fixed to 0 by the addition of Algorithm A. Similarly for \( M_0^\alpha \) be the maximum of \( L_0^\alpha \). If for all \( i \leq n = \alpha + \beta + \gamma \), \( \alpha_0(i) \leq M_0^\alpha \), \( \alpha_1(i) \leq M_1^\alpha \), \( \gamma_0(i) \leq M_0^\beta \) and \( \gamma_1(i) \leq M_1^\beta \), Algorithm A attains an exact solution. Otherwise, there exists a \( k \) such that only one of \( \alpha_0(i), \alpha_1(i), \gamma_0(i) \) and \( \gamma_1(i) \) exceeds its maximum. Without loss of generality, assume \( \alpha_1(k) = M_1^\alpha + 1 \) (other cases, proof is similar). By (13), there exist \( N_0^\alpha \) and \( N_1^\alpha \) such that

\[
N_0^\alpha + N_1^\alpha = \gamma
\]
\[
\alpha_1(k) + \beta + N_0^\alpha = C_1
\]
\[
\alpha_1(k) + N_1^\alpha = C_2
\]
\[
\alpha_0(k) + N_0^\beta = C_3 \leq D
\]
\[
(C_1, C_2) = (D, D + 1) \text{ or } (C_1, C_2) = (D + 1, D)
\]
\[
(C_1, C_2) = (D + 1, D + 1)
\]

Below we justify

for all \( i > k \), \( \alpha_1(i) = \alpha_1(k) \)

Assume the solution of the \( i \)-th (\( i > k \)) LP is

\[
L_1^\alpha + L_0^\beta = d_1
\]
\[
L_1^\alpha + L_1^\beta = d_2
\]
The addition of the 1st and 2nd equation in (7) yields

\[ L_0^\alpha + L_0^\gamma = d_3 \]

Clearly \( L_1^\alpha \geq \alpha_1(k) \), \( L_0^\alpha \leq \alpha_0(k) \) (17)

By (14), we have

\[ d_1 - d_3 = L_1^\alpha - L_0^\alpha + \beta \geq \alpha_1(k) - \alpha_0(k) + \beta = C_1 - C_3 \geq 0 \] (18)

Therefore \( d = \max(d_1, d_2, d_3) = \max(d_1, d_2) = \max(L_1^\alpha + \beta + L_0^\alpha + L_1^\gamma) \).

Namely, \( d \) decreases as \( L_1^\gamma \) decreases. Thus, by (17) we have

\[ L_1^\alpha = \alpha_1(k) \] (19)

The claim of (15) is proved. Next we shall show that

\[ \exists j > k, \alpha_1(k) + \beta + \gamma_0(i) \leq D \& \alpha_1(k) + \gamma_1(i) \leq D \] implies

\[ d = d_1 = d_2 = (C_1 + C_2)/2 \] (20)

By (16),(19),(15), we have

\[ d_1 + d_2 = 2L_1^\alpha + \beta + \gamma = 2\alpha_1(k) + \beta + \gamma = C_1 + C_2 \]

Therefore, by (18), we have

\[ d = \max(d_1, d_2) \geq (d_1 + d_2)/2 = (C_1 + C_2)/2 \] (21)

(14) can be rewritten as

\[ \alpha_1(k) + \beta + (N_0^\gamma - ((C_1 - C_2)/2)) = (C_1 + C_2)/2 \]

\[ \alpha_1(k) + (N_0^\gamma + ((C_1 - C_2)/2)) = (C_1 + C_2)/2 \] (22)

\[ \alpha_0(k) + (N_0^\gamma - ((C_1 - C_2)/2)) = C_3 - ((C_1 - C_2)/2) \]

Clearly, \( L_0^\gamma = (N_0^\gamma - ((C_1 - C_2)/2)) = 0 \) is a feasible solution of the \( i \)-th (\( i > k \)) LP, but not necessarily optimal. Therefore \( d \leq (C_1 + C_2)/2 \). By the constraint of \( C_1, C_2 \) and \( C_3 \) in (14), it is easy to verify

\[ C_3 - (C_1 - C_2)/2 \leq (C_1 + C_2)/2. \]

Thus, by (21) and (22), the claim of (20) is proved.

Below we shall prove

\[ \exists j > k \text{ s.t. } \alpha_1(k) + \beta + \gamma_0(j) = D + 1 \text{ implies } \forall i > j, \gamma_0(i) = \gamma_0(j) \] (23)

Assume \( j > k, \alpha_1(k) + \beta + \gamma_0(j) = D + 1, i > j \) (24)

Then, the \( L_0^\gamma \) of the \( i \)-th LP satisfies \( L_0^\gamma \geq \gamma_0(j) \) (25)

Then, by (19),(24) we have

\[ L_1^\alpha + \beta + L_0^\alpha \geq \alpha_1(k) + \beta + \gamma_0(j) = D + 1 \]

Thus \( d \geq D + 1 \) (26)

On the other hand, by (14), we can prove

\[ \alpha_1(k) + \beta + \gamma_0(j) = D + 1 \]

\[ \alpha_1(k) + \gamma_1(j) \leq D + 1 \]

\[ \alpha_0(k) + \gamma_0(j) \leq D + 1 \]

Therefore \( L_0^\gamma = \gamma_0(j) \) is a feasible solution of the \( i \)-th LP. It means \( d \leq D + 1 \). Thus, by (26), \( d = D + 1 \). This implies \( L_0^\gamma \leq \gamma_0(j) \). Then by (25), the claim of (23) is proven.

In a way similar to the proof of (23), we can prove

\[ \exists j > k \text{ s.t. } \alpha_1(k) + \gamma_1(j) = D + 1 \text{ implies } \forall i > j, \gamma_1(i) = \gamma_1(j) \] (27)

By (20), (23), (27) and the previous proof, we conclude that in the case \( (D_1', D_2, D_3) \neq (D, D, D - 1) \), the error of Algorithm A is at most one. Now we consider the case

\[ (D_1', D_2, D_3) = (D, D, D - 1) \]

The addition of the 1st and 2nd equation in (7) yields
\[ \beta + \gamma = D'_1 + D_2 = 2D \]  
(28)

The addition of the 1st and 2nd equation in (9) yields

\[ 2L_1^\alpha + \beta + \gamma = d_1 + d_2 \leq 2d \leq 2D \]

Then by (28), \( L_1^\alpha = 0 \). This is equivalent to \( L_0^\alpha = \alpha \). That is, without rounding error, Algorithm A fix all the letters of the \( \alpha \) substring into 0. It remains to how to compute \( L_1^\beta, L_1^\gamma \) and \( L_1^\gamma \). By symmetry, we can prove in a way similar to the previous that Algorithm A computes \( L_0^\beta, L_1^\beta, L_0^\gamma \) and \( L_1^\gamma \) within one error of optimal distance. \( \square \)

Based our empirical observation, the error caused by the algorithm was always within one. Hence, for any \( m \), the number of the input strings, we have

**Conjecture 1.** For any input, the error of Algorithm A is at most one.

### 3 Improving the running time and quality of the solution

To speed up the algorithm, we present Algorithm B, which picks multiple (not single) variables of higher values to round up at a time. That is, in the rounding phase, this algorithm searches always for multiple higher valued variables, and then set them to one’s, and the other variables at the same positions to zero’s. Selection is done by parameter \( \Theta \), which is set to 0.9 in our experiment. As long as \( x_{a,j} \geq \Theta \), we set the solution of the \( j \)-th position to \( a \).

**Algorithm B**

**Input:** \( s_1, s_2, \ldots, s_m \) and a threshold \( \Theta \geq 0.9 \)

**Output:** a center string \( t \in \Sigma \) close to every string \( s_i \)

1. for \( 1 \leq j \leq n \) do \( t[j] \leftarrow \phi \notin \Sigma \).
2. repeat the following process until all \( t[j] \neq \phi \).
   2.1 Solve the LP-relaxation
   2.2 Let \( x'_a,j \) be the value of \( x_{a,j} \) for the LP optimal solution.
      if there exists an \( x'_{a,j} \geq \Theta \)
      then for all \( x'_{a,j} \geq \Theta \) and \( t[j] = \phi \) do \( t[j] \leftarrow a \)
      else find \( x'_{b,j} \) such that \( x'_{b,j} = \max\{x'_{a,j} | a \in \Sigma, t[j] = \phi \} \)
      \( t[k] \leftarrow b \)

To get a higher precision, we improve Algorithm B by Algorithm C. It tries not only the best, but also the second best. If the first solution is not optimal, we select 8 positions to be re-solved the most possibly in the increasing order of variable values. The first position of a solution to be re-solved is one out of the 8 positions. Its value is set to the character corresponded by the second best valued variables. We update the initial setting to find a new solution. Thus, using 8 different settings, we can find 8 different solutions. Finally, we choose the best one out of 9 solutions, including the 1st solution.

**Algorithm C**

1. Let first\([k]\), second\([k]\) store the largest value of \( x \)’s variables in the \( k \)-th position, second\([k]\) the character with the second largest value.
2. Invoke Algorithm B with the following modification: the “else” statement of Algorithm B is revised as

\[
\text{find } x'_{b,j} = \max\{|x'_{a,j}| a \in \Sigma, t[j] = \phi\}
\]

\[
t[k] \leftarrow b \quad \text{first}[k] = x'_{b,j}
\]

\[
\text{second}[k] \leftarrow c \text{ with } x'_{c,j} = \max\{|x'_{a,j}| a \in \Sigma, a \neq b\}
\]

3. if the objective value of \( t = \) that of the LP rounded up, return.

else \( T \leftarrow t \)

4. for \( 1 \leq i \leq 8 \) do

   for \( 1 \leq j \leq n \) do

      \( t[j] \leftarrow \phi \)

      \( t[k] \leftarrow \text{second}[k], \text{where first}[k] \) is \( i \)-th smallest

      Use Step 2 of Algorithm B to re-solve the CSP

      if the current solution \( t \) is better than \( T \) then

      \( T \leftarrow t \)

5. \( t \leftarrow T \)

Table 1. Empirical Results for the Alphabet with 4 Characters

| Instance | Average distance | Max distance error | Average time (ms) |
|----------|------------------|--------------------|-------------------|
|          | LP Alg.C B&B IP | Alg.C B&B IP       | LP Alg.C B&B IP   |
| m | n | 10 | 300 | 175.00 | 175.00 | 175.00 | 0.80 | 0.80 | 52 | 182 | 8203 |
| 10 | 400 | 231.67 | 231.67 | 231.67 | 0.60 | 0.60 | 78 | 266 | 15271 |
| 10 | 500 | 293.00 | 293.00 | 293.00 | 0.80 | 0.80 | 114 | 349 | 25261 |
| 10 | 600 | 347.00 | 347.00 | 347.00 | 0.80 | 0.80 | 151 | 843 | 39344 |
| 10 | 700 | 409.00 | 409.00 | 409.00 | 0.60 | 0.60 | 192 | 886 | 53786 |
| 10 | 800 | 462.67 | 462.67 | 462.67 | 0.70 | 0.70 | 234 | 609 | 78167 |
| 15 | 300 | 185.33 | 185.67 | 185.67 | 1.02 | 1.02 | 104 | 375 | 342166 |
| 15 | 400 | 246.67 | 247.33 | 246.67 | 1.23 | 0.80 | 130 | 1094 | 263583 |
| 15 | 500 | 306.67 | 307.00 | 306.67 | 0.97 | 0.40 | 172 | 838 | 37786 |
| 15 | 600 | 366.67 | 367.00 | 366.67 | 1.27 | 0.46 | 229 | 1813 | 59198 |
| 15 | 700 | 426.67 | 428.67 | 428.67 | 0.97 | 0.97 | 281 | 495 | 81906 |
| 15 | 800 | 491.00 | 491.00 | 491.00 | 0.80 | 0.80 | 308 | 552 | 107703 |
| 20 | 300 | 190.67 | 191.00 | 191.00 | 1.12 | 1.12 | 130 | 880 | 344474 |
| 20 | 400 | 252.33 | 252.67 | 252.67 | 1.03 | 1.03 | 182 | 937 | 353969 |
| 20 | 500 | 315.33 | 315.33 | 315.33 | 0.59 | 0.59 | 260 | 1135 | 53875 |
| 20 | 600 | 379.67 | 380.00 | 380.00 | 1.22 | 1.22 | 312 | 1823 | 385182 |
| 20 | 700 | 443.33 | 443.33 | 443.33 | 0.73 | 0.73 | 401 | 917 | 121641 |
| 20 | 800 | 505.00 | 505.00 | 505.00 | 0.88 | 0.88 | 474 | 547 | 171245 |
| 25 | 300 | 195.00 | 196.00 | 196.00 | 1.34 | 1.34 | 151 | 1911 | 1000021 |
| 25 | 400 | 259.00 | 260.00 | 259.67 | 1.49 | 1.33 | 239 | 2729 | 694192 |
| 25 | 500 | 323.00 | 323.67 | 323.67 | 1.27 | 1.27 | 334 | 2589 | 689667 |
| 25 | 600 | 387.67 | 388.00 | 387.67 | 1.40 | 0.76 | 411 | 1817 | 113396 |
| 25 | 700 | 451.00 | 451.33 | 451.33 | 1.09 | 1.09 | 516 | 2594 | 435693 |
| 25 | 800 | 515.67 | 516.67 | 516.67 | 1.11 | 1.11 | 594 | 4776 | 1000016 |
| 30 | 300 | 197.33 | 197.67 | 197.67 | 1.26 | 1.26 | 172 | 1114 | 349266 |
| 30 | 400 | 263.00 | 263.67 | 263.33 | 1.71 | 1.02 | 276 | 2646 | 370468 |
| 30 | 500 | 328.33 | 329.00 | 328.67 | 1.63 | 1.04 | 401 | 2797 | 398458 |
| 30 | 600 | 392.67 | 393.00 | 393.33 | 1.39 | 1.54 | 516 | 4089 | 708740 |
| 30 | 700 | 459.33 | 460.00 | 459.67 | 1.57 | 1.44 | 669 | 4625 | 459099 |
| 30 | 800 | 523.00 | 523.33 | 523.67 | 1.50 | 1.52 | 740 | 5953 | 755380 |
4 Simulations

On Celeron 2.2GHz CPU, we tested two algorithms: our Algorithm C and the B&B IP by Meneses et al. which is referred to as the best IP for the CSP so far.

We carried out many experiments, including McClure data set [ML1] and random instances over the alphabet with 2 characters, 4 characters and 20 characters. In all experiments, our algorithm’s performance was very good. For the limit of space, we present only the empirical results for random instances over the alphabet with 4 characters. In Table 1, we provided three instances for each entry. Parameters $m$ and $n$ stands for the number of strings and the string size. “distance” and “time” refer to the minimum distance found, and the running time in milliseconds. LP average distance is computed as $(\lceil d_1 \rceil + \lceil d_2 \rceil + \lceil d_3 \rceil)/3$. The reason for taking the ceiling here is because the optimal solution for the CSP is no less than the ceiling of the LP value. In the 6th, 7th column, Max distance error is defined as $\max_{i=1}^{3} \{ |d_i - d_i^{LP}| \}$, where $d_i$ is the $i$-th solution, and $d_i^{LP}$ is the $i$-th LP fractional solution. The maximum time allowed for each instance was set to 1000 seconds. As was seen in Table 1, we found always an exact solution except for a few instances. In terms of running time, our improvement was huge. Our algorithm was from 32 up to 912 times faster than the B&B IP. In other experiments, which is not listed here, it was even 1765 times faster. In some cases, its speed was even close to one for computing an LP. Notice, our algorithm involves generally many LP solvers. Even so, in the worst case, it was only 20 times slower than computing an LP.

References

BL1. Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus sequences. 8th Ann. Symp. Comb. Pat. Match., LNCS 1264, 247–261(1997)
BG1. Berman, P., Gumucio, D., Hardison, R., Miller, W., Stojeanovic, N.: A linear-time algorithm for the 1-mismatch problem. Workshop Alg. & Data Stru. 125–135(1997)
CV1. Cheriyan, J., Vempala, S.: Edge covers of setpairs and the iterative rounding method. IPCO 2001, LNCS 2081, 30–44(2001)
FL1. Frances, M., Litman, A.: On covering problems of codes. Theor. Comput. Syst. 30, 113–119(1997)
GJ1. Gasieniec, L., Jansson, J., Lingas, A.: Efficient approximation algorithms for the hamming center problem. 10th ACM-SIAM Symp. Discr. Alg. 905–906(1999)
GN1. Gramm, J., Niedermeier, R., Rossmanith, P.: Exact solutions for closest string and related problems. ISAAC 2001, LNCS 2223, 441–452(2001)
JA1. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60(2001)
LL1. Lanctot, K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Information and Computation 185, 41–55(2003)
LM1. Li, M., Ma, B., Wang, L.: On the closest string and substring problems, J.ACM 49, 157–171(2002)
ML1. Meneses, C., Lu, Z., Oliveira, C., Pardalos, P.: Optimal solutions for the closest string problem via integer programming, INFORMS J. Comput. 16(4), 419–429(2004).