Journal homepage http://revistas.unitru.edu.pe/index.php/SSMM

SELECCIONES MATEMÁTICAS
Universidad Nacional de Trujillo
ISSN: 2411-1783 (Online)
2020; Vol. 7(2): 257-266.

Exact and kernelization algorithms for Closet String

Algoritmos exacto y de kernelización para el problema de la subsecuencia de caracteres más próxima

Omar Latorre Vilca

Received, Set. 10, 2020 *Accepted, Nov. 01, 2020*

How to cite this article:
Latorre Vilca O. *Exact and Kernelization algorithms for Closet String*. Selecciones Matemáticas. 2020;7(2):257–266.
http://dx.doi.org/10.17268/sel.mat.2020.02.08

Abstract

In this paper we address CLOSEST STRING problem that arises in web searching, coding theory and computational molecular biology. To solve it is to find a string that minimizes the maximum Hamming distance from a given set of strings. CLOSEST STRING is an NP-hard problem. This paper proposes two linear-time algorithms, one for the general case, a kernelization algorithm, and the other for three-strings, a linear-time algorithm called Minimization First Algorithm (MFA). A formal proof of the correctness and the computational complexity of the proposed algorithms are given.

Keywords . Closest String Problem, Combinatorial Optimization, Exact Algorithm, Fixed Parameter Algorithm, Kernelization.

Resumen

En este artículo abordamos el problema de la subsecuencia de caracteres más próxima que surge en la búsqueda web, la teoría de la codificación y la biología molecular computacional. Para resolverlo debe se encontrar una subsecuencia de caracteres que minimice la distancia de Hamming máxima de un conjunto dado de subsecuencias, dicho problema está en NP-hard. Este artículo propone dos algoritmos de tiempo lineal, uno para el caso general, un algoritmo de kernelización, y el otro para tres subsecuencias de caracteres, un algoritmo de tiempo lineal llamado Algoritmo de la primera minimización (MFA). Se expresa una prueba formal para verificar la corrección y complejidad computacional de los algoritmos propuestos.

Palabras clave. Problema de la subsecuencia de caracteres más próxima, Optimización combinatoria, Algoritmo exacto, Algoritmo de parámetro fijo, Kernelización.

1. Introduction. Let Γ be an alphabet. CLOSEST STRING or CENTER STRING problem calls for finding a string $x \in \Gamma^m$ that better approximates a given set $\mathcal{S}$ of strings $s^1, s^2, ..., s^k \in \Gamma^m$. Approximation is measured with the Hamming distance $d(x, y)$, that counts the number of different characters in x and y. An optimal solution of CLOSEST STRING is an x^* that, among all strings $x \in \Gamma^m$, minimizes the maximum distance $d(x, s^i)$ from any $s^i \in \mathcal{S}$. It is NP-complete (Frances and Litman, [5]).

In computational biology, problems related with strings often arise: given strings are compared with each other and their common part is searched for. In cryptography for data compression, one faces a similar challenge (Storer, [11]). One of the challenges of web searching is multiple occurrences of the same data, whether in exact duplicates or with minor changes. For the same reasons there is a greater practical interest in finding methods to solve it.

Several fixed-parameter solutions were given for CLOSEST STRING, such that, (Gramm *et al.*, [7]) in 2003 proposed the first fixed-parameter algorithm running in $\mathcal{O}(km + kd^{d+1})$ time for finding a string x such that $\max_{1 \le i \le k} d(x, s^i) \le d$. (Ma and Sun, [10]) presented another algorithm running in $\mathcal{O}(km +$

*Facultad de Ciencias, Universidade Estadual de Mato Grosso do Sul, Cidade Universitária de Dourados, Mato Grosso do Sul-Brazil (omarlatorre@comp.uems.br).

$kd(16|\Gamma|)^d)$ time, where Γ denotes the alphabet. (Gramm *et al.*, [6]) in 2001 proposed a recursive algorithm for finding a consensus string x for three strings, but its result is only for theoretical interest. (Liu *et al.*, [9]) presented a linear-time algorithm for 3-strings with a binary alphabet, (Boucher *et al.*, [3]) proposed a linear algorithm for finding a string x such that $max_{1,\dots,4}d(x, s^i) \leq d$ for four binary strings, and (Amir *et al.*, [1]) in 2016 posed a quadratic time algorithm for 5 strings with a binary alphabet.

Table 1.1: A list of articles that uses exact methods to solve Closest String.

Year	#Strings	Alphabet	Running time	References
2003	k	q	$\mathcal{O}(km + kd^{d+1})$	Gramm *et al.*, [7]
2008	k	q	$\mathcal{O}(km + kd(16\|\Gamma\|)^d)$	Ma and Sun, [10]
2001	3	q	$\mathcal{O}(m)$	Gramm *et al.*, [6]
2011	3	2	$\mathcal{O}(m)$	Liu *et al.*, [9]
2009	4	2	$\mathcal{O}(m)$	Boucher *et al.*, [3]
2016	5	2	$\mathcal{O}(m^2)$	Amir *et al.*, [1]

(Lenstra, [8]) in 1983 showed that Integer Linear Programming (ILP) with a fixed number of variables can be solved with $O(p^{9p/2}L)$ arithmetic operations with integers of $O(p^{2p}L)$ bits in size, where p is the number of ILP variables and L is the number of bits in the input. Using this famous result, in 2011, (Gramm, Niedermeier, and Rossmanith, [7]) presented an ILP formulation for CLOSEST STRING where the number of variables is $O(k! \times k)$, thus providing a Fixed-Parameter Tractable (FPT) algorithm for CLOSEST STRING parameterized by k (number of input-strings).

A fundamental and very powerful technique in designing FPT algorithms is kernelization. In a nutshell, a kernelization algorithm for a parameterized problem is a polynomial-time transformation that transforms any given instance to an equivalent instance of the same problem, with size and parameter bounded by a function of the parameter in the input. Typically this is done using so-called reduction rules, which allow the safe reduction of the instance to an equivalent 'smaller' instance.

Related to kernelization algorithms, (Gramm *et al.*, [6]) showed that CLOSEST STRING is FPT parameterized by k (number of strings), they also proved a polynomial time reduction $\mathcal{O}(k^2 d \log k)$. Also, (Basavaraju *et al.*, [2]) presented that CLOSEST STRING is not likely to have polynomial kernels when parameterized by d (Hamming distance), they arrived at the results by showing a polynomial parameter transformation from CNF-SAT parameterized by the number of variables.

Notation. Throughout this paper, we will be considering our input-instance as a matrix. If the set $\mathcal{S}$ has k strings, $s^1, \dots, s^k$, each of length m, we view $\mathcal{S}$ as a $k \times m$ matrix. We can thus refer to columns and rows. We will say column to denote a column of this matrix, and row to denote a row in the matrix. Thus, e.g., the element is the second row of column j, is the jth symbol of s^2. The distance of a string s' from $\mathcal{S}$ is $max_{s \in \mathcal{S}} d(s', s)$. Given a string s we denote with $\sigma_j(s)$ the character in the jth position of s. We refer to two identical columns as having the same column type also known as tuple. We denote with $T(\mathcal{S})$ the column types of $\mathcal{S}$. For a column type $t \in T(\mathcal{S})$ we denote with $\#t$ the number of column types t in S. A set of column positions $\mathcal{I}$ is denoted as $\mathcal{I} = \{p_1, \dots, p_m\} \subseteq \{1, \dots, m\}$ where $p \in \mathcal{I}$ is a specific column position. Given a string s we denote with $s_{\mathcal{P}}$ a substring of s restricted to a set of indices $\mathcal{P}$.

In the following, the article is divided in sections. Section 2 presents an integer linear programming (ILP) formulation for CLOSEST STRING. Section 3 poses a kernelization algorithm. Section 4 shows an ILP formulation for 3-strings, obtained using the ILP presented in Section 2. Section 5 claims a linear-time algorithm for 3-strings, and at last, the conclusion section concludes the paper.

2. An ILP formulation for CLOSEST STRING.

(Gramm *et al.*, [7]) suggested an ILP-formulation based on column types. This yields Fixed-Parameter Tractable algorithm for CLOSEST STRING parameterized by k (number of input strings):

$$\min \quad d; \tag{2.1}$$

$$\text{s.t.:} \quad \sum_{t \in T(S)} \sum_{\sigma \in \Gamma \setminus \{\sigma_{t,i}\}} x_{t,\sigma} \leq d, \qquad \forall i \in \{1, \dots, k\}, \tag{2.2}$$

$$\sum_{\sigma \in \Gamma} x_{t,\sigma} = \#t, \qquad \forall t \in T(S), \tag{2.3}$$

$$x_{t,\sigma} \in \{0, 1, \dots, \#t\}, \qquad \forall \sigma \in \Gamma, \forall t \in T(S). \tag{2.4}$$

In this formulation, for every $t \in T(S)$ and $\sigma \in \Gamma$, the variable $x_{t,\sigma}$ is the number of occurrences that σ has in the closest string at locations that correspond to column type t. Let $s^i \in S$ and let $\sigma_{t,i}$ be the character of string s^i in column type t. The restrictions (2.2) calculate the Hamming distance of s^i from the center string. That is, for each column type t we sum the characters in the center string at locations that correspond to t that are different from the character s^i has in these locations. Constraints (2.3) impose the number of column types existed in each tuple, and (2.4) makes $x_{t,\sigma}$ to have integer values. Finally, the objective is minimized the value d.

3. Kernelization algorithm for Closest String. A folk theorem in the Parameterized Complexity asserts that a decidable problem is fixed-parameter tractable if and only if it admits a kernelization algorithm. Clearly a kernelizable decidable problem can be solved in FPT-time by applying, after getting the kernel, an algorithm that proves that the problem is decidable. On the other hand, from an algorithm that runs in at most $f(k) \times n^c$ steps (Basavaraju *et al.*, [2]), we can obtain a kernel of size $f(k)$ by applying the first n^{c+1} steps of the algorithm. If it terminates with an answer, it is done; otherwise, the instance size is bounded by $f(k)$ and then it is a kernel (Fomin *et al.*, [4]).

We first observe that considering the ILP formulation proposed for CLOSEST STRING and using the previously described approach (see Section 2), we are able to get a kernel of size $O((k! \times k)^{9(k! \times k)/2})$ for the problem. Thus, our goal in this section is present a linear kernelization algorithm which returns a kernel smaller that the previous one. To do that, we introduce one lemma, then use it to prove that Kernelization algorithm can find a kernel for the general case in linear time when k (number of input-strings) is fixed.

Theorem 3.1. *Closest String admits a linear kernelization algorithm which obtains a kernel of size* $O(k!^2)$.

Proof. As in (Gramm *et al.*, [7]), we introduce some preliminary concepts. Given a set of k strings of length m, we interpret these input strings as a $k \times m$ character matrix M. As first observed in (Gramm *et al.*, [7]), after reordering the columns of a CLOSEST STRING instance, it is easy to obtain solutions for the original instance from solutions of the reordered instance. In fact, the maximum distance from the optimal solution to the input-instances in both original and reordered instances are equal. Another observation in (Gramm *et al.*, [7]) is that the columns are independent from each other in the sense that the distance from the closest string is measured columnwise. Thus, we can work with only normalized strings. A string is said to be normalized when 'a' denotes the letter that occurs most often in a column, 'b' always denotes the letter that has the second most frequent occurrences and so on. Therefore, the first steps of our kernelization are the following:

(i) if $m \leq k!$ then the input strings already form a kernel. Otherwise,
(ii) normalized the input strings;
(iii) sort the columns of the CLOSEST STRING instance.

The second preprocessing can be easily done in linear time. In addition, as $m \geq k!$, using a stable sorting such as counting sort we can also perform the third preprocessing in linear time. Now, the instance is normalized and has at most $k!$ distinct column types. At this point it is intuitive to imagine that a certain number of columns from any column type do not make the problem more difficult. So we have to determine a sufficient number of column types to be added in the kernel to be built.

Definition 3.1. *Let* $s_1, s_2, \ldots, s_k$ *be a set of strings forming a* $k \times m$ *character matrix* M *such that all columns have the same column type* t_j. M *is said a* good block *if* $0 \equiv m \mod c_j$ *where* c_j *is the number of distinct symbols of the column type* t_j.

Claim 1. Let $s_1, s_2, \ldots, s_k$ be a set of strings forming one good block. The closest string for such a set is a *regular solution*, i.e., it has $\frac{m}{c_j}$ occurrences of each symbol in t_j.

Claim 2. Let $s_1, s_2, \ldots, s_k$ be a set of strings forming exactly a set of good blocks. The closest string for such a set is obtained by concatenation of the regular solution of each block.

Now, we have more rules for our kernelization:

(iv) if the input strings exactly form a set of good blocks then solve the problem in linear time. Otherwise,
(v) for each column type t_j remove $m \mod c_j$ column types and add them to the kernel Q to be constructed.

Note that, $m \mod c_j < k$, for every column type. Thus, the character matrix Q has at most $k! \times k - 1$ columns, and M now exactly forms a set of good blocks. We denote by *irregular* a solution that is not regular, and by $Q \odot M$ the concatenation of Q and M.

Let A and B be two partially filled matrices with the same dimensions and non-overlapping sets of column-positions, consider two distinct rows $s \in A$ and $s' \in B$ we have (if $s[p] = \llcorner\!\lrcorner$, then $s'[p] \neq \llcorner\!\lrcorner$ & vice versa), that is, $\{s, s'\} \in (\Gamma \cup \llcorner\!\lrcorner)^m$, the output of the function $A \odot B$ is a character matrix, more formally we have:

$$A \odot B = \begin{cases} s[p] & \text{if } s'[p] = \text{␣} \qquad p = 1, \dots, m; \\ s'[p] & \text{if } s[p] = \text{␣} \qquad \forall s \in A, \forall s' \in B. \end{cases} \tag{3.1}$$

Although counter-intuitive, the optimal solution for Closest String in $Q \odot M$ is not necessarily obtained by simple concatenation of the optimal solutions of Q and M. By fixing a solution to Q that can be extended in an optimal solution to $Q \odot M$, sometimes the local solution to some minimal good block of M may not be an optimal local solution, since the solution of Q provides different distance to the input strings. In other words, the irregularity of a solution for a minimal good block can fit in a complementary way with the irregularity of a solution of Q in order to obtain a solution that minimize the maximum distance. Thus, we are required to determine an upper bound for the number of minimal good blocks of M since we can have an irregular solution in the closest string of $Q \odot M$.

Lemma 3.1. *At most $k! \times (k^3 - 2k^2 + k)$ minimal good blocks of the same column type will have irregular partial solution in the optimal solution of $Q \odot M$.*

Proof: In the worst case, the local solution of each good block will differ from a regular solution in just one symbol. For a fixed solution s_Q of Q, let $d(s_Q, \ell)$ be the distance between s_Q and the string in row ℓ of Q. For two distinct rows ℓ and r let $d_{s_Q}(\ell, r) = |d(s_q, \ell) - d(s_q, r)|$. (a) For a given column type t_j, if column types can be added to Q in order to minimize $d_{s_Q}(\ell, r)$ then by adding to Q some column types (at most $k-1$), it is possible to extend s_Q decrementing in one unit $d_{s_Q}(\ell, r)$. Let $d_{\max} = \max d_{s_Q}(\ell, r)$. (b) $d_{\max} \leq k! - k - 1$. As Q has k rows then by (a) and (b) we have that at most $k! \times k^3 - 2k^2 + k$ blocks will have irregular partial solution in the optimal solution of $Q \odot M$.

Given Lemma 3.1, we have the following reduction rules:

(vi) for each column type t_j in M, remove $k! \times (k^3 - 2k^2 + k)$ minimal good blocks of M and add it into Q.

As any minimal good block has size at most k, Q is a kernel size at most $k! \times (k-1) + k! \times (k^4 - 2k^3 + k^2)$ where the solution of the original instance can be obtained using the closest string of Q and a regular solution for each maximal good block that is not in Q, which conclude the proof of Theorem 3.1. □

Combining this normalized kernel of size $O(k!^2)$, with a brute force algorithm we can perform an algorithm to solve Closest String in $O(k^{k!^2} \times m)$ time, which asymptotically already gives us a great improvement when compared to the ILP approach.

4. An ILP formulation for 3-strings. (Gramm *et al.*, [6]) solved the decision version of this ILP directly using the algorithm of (Lenstra, [8]) that has an exponential dependency in the number of variables. Thus, they were not able to solve the ILP for more than four strings. The authors suggested an ILP-formulation based on column types. This yields fixed-parameter tractability for CLOSEST STRING with respect to parameter k (number of strings).

Define n_0 through n_4 as the number of column types v_0 through v_4. In column type v_0, you clearly always pick α. Otherwise, you need to assign $x_{\alpha,1}$ of the column type v_1 to α (and the rest, $x_{\beta,1} = n_1 - x_{\alpha,1}$, to β), $x_{\alpha,2}$ column type v_2 to α (and the rest, $x_{\beta,2} = n_2 - x_{\alpha,2}$ to β), $x_{\alpha,3}$ column type v_3 to α (and the rest, $x_{\beta,3} = n_3 - x_{\alpha,3}$ to β), and $x_{\alpha,4}, n_{\beta,4}, x_{\gamma,4}$ column type v_4 to α, β, γ respectively (with $x_{\alpha,4} + x_{\beta,4} + x_{\gamma,4} = n_4$). Clearly, given $x_{\alpha,1}$ through $x_{\gamma,4}$, the process of "normalize the columns, reorder the strings if necessary, make the selections, un-normalize the columns" takes time $O(m)$(Gramm *et al.*, [7]). The numbers $x_{\alpha,1}$ through $x_{\gamma,4}$ are the solution to the following ILP:

$$\begin{aligned} \min \quad & d \\ s.t.: \quad & x_{\alpha,1} + x_{\beta,2} + x_{\beta,3} + x_{\beta,4} + x_{\gamma,4} \leq d, \\ & x_{\beta,1} + x_{\alpha,2} + x_{\beta,3} + x_{\alpha,4} + x_{\gamma,4} \leq d, \qquad (4.1) \\ & x_{\beta,1} + x_{\beta,2} + x_{\alpha,3} + x_{\alpha,4} + x_{\beta,4} \leq d, \\ & \sum_{\sigma \in v_j} x_{\sigma,j} = n_j \quad j = 1, \dots, 4, \qquad (4.2) \\ & x_{\sigma,j} \in \{0, 1, \dots, n_j\}, \qquad (4.3) \\ & d \in \mathbb{Z}_+. \qquad (4.4) \end{aligned}$$

In this formulation, for every column position type $v_j \in V(S)$ where $j = 1, \dots, 4$ and $\sigma \in v_j$, the variable $x_{\sigma,j}$ is the number of occurrences that σ has in the closest string at locations that correspond to column type v_j. Let $s^i \in S$ where $i = 1, \dots, 3$ be the input-instance, and let σ be the character of string

s^i in column-position type v_j. The restrictions (4.1) calculate the Hamming distance of s^i from the center string, that is, for each column type v_j we sum the characters in the center string at locations that correspond to v_j that are different from what the character s^i has in these locations. Constraints (4.2) impose the number of column types existed in each tuple, and (4.3) makes $x_{\sigma,j}$ have integer values. Finally, the objective is to minimize the value d.

5. Linear-time algorithm for 3-strings. In the Minimization First Algorithm (MFA) (Vilca, [12]), the identification of column types is needed. The algorithm breaks up into five column types. Finally, it decides a character for each column type by simple evaluation through the different cases according to the number of column types presented in the input instance. MFA can find the optimal solution by traversing all the positions of the input strings only once. With the algorithm we obtain, the optimal value $\lceil max_{i,j=1,2,3} d(s^i, s^j)/2\rceil$ and when the number of all "mismatches" tuples is greater than the others, its optimal value is $\lceil (n_1 + n_2 + n_3 + 2n_4)/3\rceil$.

```
Input: $\mathcal{S} = \{s^1, s^2, s^3\}$: a 3-strings instance with $m$ the length of strings.
Output: $x$: optimal solution such that $d(x, s) \leq d_{opt}\ \forall s \in \mathcal{S}$.
if $|\mathcal{S}| = 3$ then
    Sort the pairwise distances of $s^1$, $s^2$ and $s^3$.
    Let $u_1$,$u_2$ and $u_3$ satisfy $d(s^{u_1}, s^{u_2}) \geq d(s^{u_1}, s^{u_3}) \geq d(s^{u_2}, s^{u_3})$
    $k_1 \leftarrow d(s^{u_1}, s^{u_2}) - d(s^{u_2}, s^{u_3})$
    $k_2 \leftarrow d(s^{u_1}, s^{u_2}) - d(s^{u_1}, s^{u_3})$
    $k_3 \leftarrow 0$ ;                          // it iterates over the 3-strings
    Let $n_4$ be the number of column positions when all mismatches
    if $n_4 = 0$ then
        count $\leftarrow \lfloor (d(s^{u_1}, s^{u_3}) - d(s^{u_2}, s^{u_3}))/2\rfloor$ ;       // Case 1
    end
    else if $k_1 + k_2 > n_4$ then
        if $k_2 > n_4$ then
            count $\leftarrow \lfloor (k_1 - k_2 + n_4)/2\rfloor$; $k_1 \leftarrow 0$; $k_2 \leftarrow n_4$ ;   // Case 3
        end
        else
            count $\leftarrow \lfloor (k_1 + k_2 - n_4)/2\rfloor$; $k_1 \leftarrow n_4 - k_2$ ;   // Case 4
        end
    end
    else
        count $\leftarrow 0$ ;                                           // Case 2
    end
    for $i$=1 to $m$ do
        if $(s_i^{u_1} \neq s_i^{u_3}$ and $s_i^{u_1} \neq s_i^{u_2}$ and $s_i^{u_2} \neq s_i^{u_3})$ then
            if $(k_1 > 0$ or $k_2 > 0)$ then
                if $(k_1 > 0)$ then
                    $x_i \leftarrow s_i^{u_1}$; $k_1 \leftarrow k_1 - 1$
                end
                else if $(k_2 > 0)$ then
                    $x_i \leftarrow s_i^{u_2}$; $k_2 \leftarrow k_2 - 1$
                end
            end
            else
                $x_i \leftarrow s_i^{u_{k_3+1}}$; $k_3 \leftarrow (k_3 + 1)\%3$
            end
        end
        else if $(s_i^{u_1} = s_i^{u_2}$ or $(s_i^{u_1} \neq s_i^{u_3}$ and count $> 0))$ then
            $x_i \leftarrow s_i^{u_1}$
            if $(s_i^{u_1} \neq s_i^{u_3}$ and $s_i^{u_2} = s_i^{u_3})$ then
                count $\leftarrow$ count $-1$
            end
        end
        else
            $x_i \leftarrow s_i^{u_3}$
        end
    end
end
```

Algorithm 1: MFA pseudo-code.

In this algorithm w.l.o.g. consider the pairwise distances of strings s^1, s^2 and s^3 satisfy $d(s^1, s^2) \geq$

$d(s^1, s^3) \geq d(s^2, s^3)$, that is, string s^1 is farthest and s^3 is closest to the other two strings. In Algorithm 1, if the number of "all mismatches" tuples is zero, we have the binary case, then count is $\lfloor (d(s^{u_1}, s^{u_3}) - d(s^{u_2}, s^{u_3}))/2 \rfloor$. Otherwise, if $k_1 + k_2 > n_4$ and $k_2 > n_4$, then k_1 is 0, k_2 is n_4, and count is $\lfloor (k_1 - k_2 + n_4)/2 \rfloor$; if $k_1 + k_2 > n_4$ and $k_2 \leq n_4$, then k_1 is $n_4 - k_2$, and count is $\lfloor (k_1 + k_2 - n_4)/2 \rfloor$; finally, if $k_1 + k_2 < n_4$, then count is zero.

Algorithm 1 assigns the characters of s^1 to the solution x among the positions where (all mismatches but $k_1 > 0$) or (s^1 matches s^2 or s^1 mismatches s^3 but count > 0), and once count is reduced to zero, it fixes the characters of s^3 in the solution x. It assigns the characters of s^2 to the solution x among the positions where all mismatches but $k_2 > 0$. Finally once k_1 and k_2 are reduced to zero, it fixes the characters of the set of strings in a round-robin order in the solution x among the positions where all mismatches.

5.1. Running time. The if-then conditional of lines 1-31 is executed when the input-instance has 3-sequences; this conditional block requires 1 step. Lines 2 and 3 sort the pairwise distances from s^1, s^2, and s^3, it requires $4m$ iterations to get the Hamming distances and c_1 steps to sort them in non-decreasing order by its Hamming distances which is constant time. Lines 4 and 5 make arithmetic operations over the Hamming distances; those operations take a constant time c_2. In Line 7, the n_4 variable counts the number of times that the column-position type v_4 (all mismatches) are presented in the input-instance, it takes m steps. The if-then-else conditional of lines 8-16 makes arithmetic operations, it takes a constant time c_3. The for-loop of lines 17-31 iterates in m steps, so it requires $3m$ iterations. The if-then-else conditional of lines 18-31 makes arithmetic operations, it takes a constant time c_4. Therefore, the running time of MFA is proven to be $O(8m)$.

5.2. Theoretical analysis. We first introduce three lemmas, then use them to prove that MFA can find an optimal solution of 3-strings with an arbitrary alphabet.

Lemma 5.1. *Let $\mathcal{S} = \{s^1, s^2, s^3\}$ be a CLOSEST STRING instance with 3-strings and length m. If x a string is an optimal solution of CLOSEST STRING instance and d_{opt} is the corresponding distance, then $\lceil (n_1 + n_2 + n_3 + 2n_4)/3 \rceil \leq d_{opt}$.*

Proof: Independently of whether the character appears in the column position j in the string solution x, it mismatches with minimum 1 character for v_1, v_2, v_3 and with minimum in 2 characters for v_4. The sum of these values is equal to the Hamming distance between x and $s^i \in \mathcal{S}$; dividing it by 3 we get the average Hamming distance.

Lemma 5.2 (Liu *et al.*, [9]). *Let S be an instance of CLOSEST STRING. If x a string is an optimal solution of CLOSEST STRING and d_{opt} is the corresponding distance, then $d_{opt} \geq \lceil max_{i,j=1,\ldots,k} d(s^i, s^j)/2 \rceil$.*

Lemma 5.3. *Let $\mathcal{S} = \{s^1, s^2, s^3\}$ be a CLOSEST STRING instance with 3-strings and length m. The number of columns satisfies the restriction $n_1 \geq n_2 \geq n_3$.*

Proof: From (5.1)-(5.5), we have $d(s^1, s^2) = n_1 + n_2 + n_4$, $d(s^1, s^3) = n_1 + n_3 + n_4$, and $d(s^2, s^3) = n_2 + n_3 + n_4$. Assume without loss of generality that $d(s^1, s^2) \geq d(s^1, s^3) \geq d(s^2, s^3)$. After making arithmetic operations, we have, $n_2 \geq n_3$ and $n_1 \geq n_2$. Finally, we get, $n_1 \geq n_2 \geq n_3$.

Theorem 5.1. *Given a CLOSEST STRING instance with 3-strings and an arbitrary alphabet. MFA always finds an optimal solution value.*

Proof: The proof is composed by four cases, it is made by direct method.

(5.1)	v_0	$s^1_j = s^2_j = s^3_j$	all matches,
(5.2)	v_1	$s^1_j \neq s^2_j = s^3_j$	s^1_j is the minority,
(5.3)	v_2	$s^2_j \neq s^1_j = s^3_j$	s^2_j is the minority,
(5.4)	v_3	$s^3_j \neq s^1_j = s^2_j$	s^3_j is the minority,
(5.5)	v_4	$s^1_j \neq s^2_j, s^1_j \neq s^3_j$, and $s^2_j \neq s^3_j$	all mismatches.

From (5.1)-(5.5), we have the identification of column types, also called tuples. Observe that, some columns presented in the input instance are repeated, therefore, the number of different tuples presented in any input-instance is equals to five.

Consider the alignment of the three strings s^1, s^2, and s^3. Note that, from (5.1)-(5.5), we have, in general, for any input-instance there are five different column types. According to Lemma 5.3, we get $n_1 \geq n_2 \geq n_3$. It follows that:

Case 1. If $n_4 = 0$, we have the binary case [Liu *et al.*, [9] proved it]. Then in Algorithm 1, the initial value of variable **count** is equal to $\lceil (n_1 - n_2)/2 \rceil$. Note that $|\Gamma| = 2$ and so either $s^3_j = s^1_j$ or $s^3_j = s^2_j$ among the positions where s^1 mismatches s^2. Hence $d(s^1, s^2) = n_1 + n_2$. The solution x of MFA is decided by :

(1.1) Among the positions where s^1 matches s^2, $x_j = s^1_j = s^2_j$.

(1.2) Among the positions where s^1 mismatches s^2:

$$(5.6)\qquad \begin{aligned} d(x,s^1) &\leq n_1 - \lceil (n_1-n_2)/2 \rceil, \\ d(x,s^2) &\leq n_2 + \lceil (n_1-n_2)/2 \rceil, \\ d(x,s^3) &\leq n_3 + \lceil (n_1-n_2)/2 \rceil \leq n_3 + \lceil (d(s^1,s^2)-2n_2)/2 \rceil \\ &\leq n_3 - n_2 + \lceil d(s^1,s^2)/2 \rceil \leq \lceil d(s^1,s^2)/2 \rceil. \end{aligned}$$

Taken together, sub-cases (1.1) and (1.2) give a lower bound (See Figure 5.1(a)), that is,

$$(5.7)\qquad max_{i=1,2,3} d(x,s^i) \leq d(s^1,s^2)/2.$$

Case 2. If $n_1 - n_3 + n_2 - n_3 \leq n_4$. Then in Algorithm 1, the initial value of variable **count** is equal to $0,\ k_1 = n_1 - n_3,\ k_2 = n_2 - n_3$. Note that $|\Gamma| = 3$. The solution x of MFA is decided as:

(2.1) Among the positions where there are at least two different symbols, we have

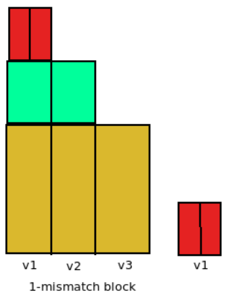
$$(5.8)\qquad x_j = argmax_{\sigma \in \Gamma} \sum_{i=1}^{3} |s_j^i = \sigma|.$$

(2.2) Among the positions where all characters mismatches:

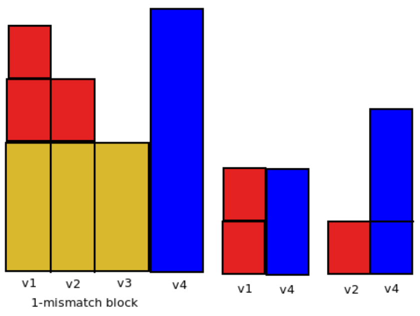
$$(5.9)\qquad \begin{aligned} d(x,s^1) &\leq n_3 + k_1 + k_2 + \lceil 2(n_4 - k_1 - k_2)/3 \rceil, \\ d(x,s^2) &\leq n_3 + k_1 + k_2 + \lceil 2(n_4 - k_1 - k_2)/3 \rceil, \\ d(x,s^3) &\leq n_3 + k_1 + k_2 + \lceil 2(n_4 - k_1 - k_2)/3 \rceil, \\ &\leq n_1 + n_2 - n_3 + \lceil 2(n_4 - n_1 - n_2 + 2n_3)/3 \rceil, \\ &\leq \lceil (n_1 + n_2 + n_3 + 2n_4)/3 \rceil. \end{aligned}$$

From (2.1) and (2.2), and by the Lemma 5.1 (See Figure 5.1(b)), we obtain,

$$(5.10)\qquad max_{i=1,2,3} d(x,s^i) \leq \lceil (n_1+n_2+n_3+2n_4)/3 \rceil \leq d_{opt}.$$



(a) Case 1



(b) Case 2

Figure 5.1: Illustration of application of Case 1, CLOSEST STRING instance with 3-strings and a binary alphabet. There are three column types to be considered v_1, v_2, and v_3. The algorithm fixes 1-mismatch blocks, in goldenrod orange and green colors, by their majority consensus value; after that, it assigns column type v_1 (in red color), half of them by their majority and for other one by their minority consensus value. Illustration of application of Case 2, there are four column types to be considered v_1, v_2, v_3, and v_4.The algorithm fixes 1-mismatch blocks, in goldenrod orange color, by their majority consensus value, after that, it assigns v_1 and v_4 column types by their majority consensus value and the character of s^1; it fixes v_2 and v_4 column types by their majority consensus value and the character of s^2; finally it assigns the rest of column types v_4 in a round-robin order.

Case 3. If $n_1 - n_3 + n_2 - n_3 > n_4$ and $n_2 - n_3 > n_4$. Then in Algorithm 1, the initial value of variable **count** is equal to $\lceil (n_1 - n_3 - (n_2 - n_3 - n_4))/2 \rceil,\ k_1 = 0,\ k_2 = n_4$. Note that $|\Gamma| = 3$ and so either $s_j^3 = s_j^1$ or $s_j^3 = s_j^2$ among the positions where s^1 mismatches s^2, $d(s^1,s^2) = n_1 + n_2 + n_4$. The solution x of MFA is decided as:

(3.1) Among the positions where s^1 matches s^2, $x_j = s^1_j = s^2_j$.
(3.2) Among the positions where s^1 mismatches s^2:

$$\begin{aligned} d(x,s^1) &\leq n_1 - \lceil (n_1 - n_3 - (n_2 - n_3 - n_4))/2 \rceil + n_4, \\ d(x,s^2) &\leq n_2 + \lceil (n_1 - n_3 - (n_2 - n_3 - n_4))/2 \rceil, \\ d(x,s^3) &\leq n_3 + \lceil (n_1 - n_3 - (n_2 - n_3 - n_4))/2 \rceil + n_4 \\ &\leq n_3 + \lceil (n_1 - n_2 + n_4)/2 \rceil + n_4 \\ &\leq \lceil (n_1 + n_2 + n_4)/2 \rceil. \\ \text{Since } k_2 &= n_2 - n_3 = n_4 \text{ we get } n_3 = n_2 - n_4. \end{aligned} \tag{5.11}$$

From (3.1) and (3.2), and by the Lemma 5.2 (See Figure 5.2(a)) we get,

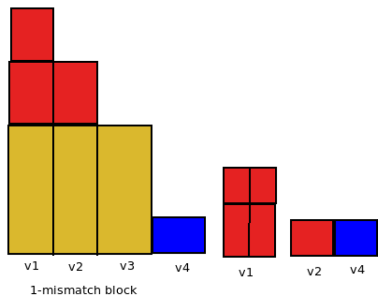
$$max_{i=1,2,3} d(x,s^i) \leq d(s^1,s^2)/2 \leq d_{opt}. \tag{5.12}$$

Case 4. If $n_1 - n_3 + n_2 - n_3 > n_4$ and $n_2 - n_3 \leq n_4$. Then in Algorithm 1, the initial value of variable **count** is equal to $\lceil (n_1 - n_3 - (n_4 - n_2 + n_3))/2 \rceil$, $k_1 = n_4 - n_2 + n_3$, $k_2 = n_2 - n_3$. Note that $|\Gamma| = 3$ and so either $s^3_p = s^1_j$ or $s^3_j = s^2_j$ among the positions where s^1 mismatches s^2, $d(s^1,s^2) = n_1 + n_2 + n_4$. The solution x is decided as:
(4.1) Among the positions where s^1 matches s^2, $x_j = s^1_j = s^2_j$.
(4.2) Among the positions where s^1 mismatches s^2:

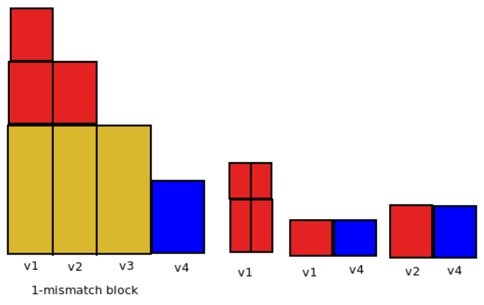
$$\begin{aligned} d(x,s^1) &\leq n_1 - \lceil (n_1 - n_3 - (n_4 - n_2 + n_3))/2 \rceil + n_2 - n_3, \\ d(x,s^2) &\leq n_2 + \lceil (n_1 - n_3 - (n_4 - n_2 + n_3))/2 \rceil + n_4 - n_2 + n_3, \\ d(x,s^3) &\leq n_3 + \lceil (n_1 - n_3 - (n_4 - n_2 + n_3))/2 \rceil + n_4, \\ &\leq \lceil (n_1 + n_2 + n_4)/2 \rceil. \end{aligned} \tag{5.13}$$

Both (4.1) and (4.2), and by the Lemma 5.2 (See Figure 5.2(b)), we obtain,

$$max_{i=1,2,3} d(x,s^i) \leq d(s^1,s^2)/2 \leq d_{opt}. \tag{5.14}$$



(a) Case 3



(b) Case 4

Figure 5.2: Illustration of application of Case 3. There are four column types to be considered v_1, v_2, v_3, and v_4.The algorithm fixes 1-mismatch blocks, in goldenrod orange color, by their majority consensus value; after that, it assigns v_2 and v_4 column types by their majority consensus value and the character of s^2; it fixes column type v_1 (in red color), half of them by their majority and for other one by their minority consensus value. Illustration of application of Case 4, there are four column types to be considered v_1, v_2, v_3, and v_4.The algorithm fixes 1-mismatch blocks, in goldenrod orange color, by their majority consensus value, after that, it assigns v_1 and v_4 column types by their majority consensus value and the character of s^1; it fixes v_2 and v_4 column types by their majority consensus value and the character of s^2; finally it fixes column type v_1 (in red color), half of them by their majority and for other one by their minority consensus value.

Altogether Cases 1, 3, and 4 prove that the solution x of MFA is an optimal one and the optimal solution is $\lceil max_{i,l=1,2,3} d(s^i,s^l)/2 \rceil$. Also, the optimal solution for Case 2 is $\lceil (n_1 + n_2 + n_3 + 2n_4)/3 \rceil$. Thus the theorem holds.

Example 5.1. *Let $\mathcal{S}$ be a CLOSEST STRING instance with 3-strings and each string of length 10, $\mathcal{S}'$ is obtained from $\mathcal{S}$ where the set of strings was ordered in a pairwise manner by their Hamming distances, we get:*

$$\mathcal{S} = \begin{cases} AGTATTGGTG \\ CCCTTTGAGA \\ TAGTGGGTCT \end{cases}, \qquad \mathcal{S}' = \begin{cases} TAGTGGGTCT \\ AGTATTGGTG \\ CCCTTTGAGA \end{cases},$$

$$\mathcal{S}' = \left\{ \begin{array}{ccc|cccc|ccc} v_4 & v_4 & v_4 & v_2 & v_1 & v_1 & v_0 & v_4 & v_4 & v_4 \\ \hline \mathbf{T} & \mathbf{A} & G & \mathbf{T} & G & G & \mathbf{G} & \mathbf{T} & C & T \\ A & G & \mathbf{T} & A & \mathbf{T} & \mathbf{T} & \mathbf{G} & G & \mathbf{T} & G \\ C & C & C & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{G} & A & G & \mathbf{A} \\ \hline T & A & T & T & T & T & G & T & T & A \end{array} \right. .$$

The number of columns of "all mismatches" column type is $n_4 = 6$. The three counters have the following values $k_1 = n_1 - n_3 = 2$, $k_2 = n_2 - n_3 = 1$, and count $= 0$. With these values, MFA identifies the Case 2, thus the optimal solution for S' or S is x = TATTTGTTA with Hamming distance $d(x, s) \leq 5, \quad \forall s \in \mathcal{S}$. Note that the optimal solution value for the Case 2 is $\lceil (n_1+n_2+n_3+2n_4)/3 \rceil = \lceil (2+1+0+2(6))/3 \rceil = 5$.

6. Conclusions.

1. In this work we show a Kernelization algorithm for the general case and its proof of correctness; we can reduce an initial kernel of size $O((k! \times k)^{9(k! \times k)/2})$ to a lesser kernel of size $O(k!^2)$. Furthermore our proposed algorithm finds a kernel in linear time, meanwhile the ILP approach runs in polynomial time, that is, $\mathcal{O}(k^2 d \log k)$.
2. We pose a linear-time algorithm for CLOSEST STRING with three strings for an arbitrary alphabet; it is mainly a column type characterization. This is an important contribution since there was only proof of polynomiality, without a polynomial algorithm actually being presented in the literature. Our approach can be extended to a major instances for example for $k > 3$ with a binary alphabet.
3. For future works, it remains an open problem to find a consensus for $k \geq 4$ strings. This problem does not look easy even for four strings.

7. Acknowledgements. This work was supported by the Brazilian Federal Agency for Post-Graduate Education (CAPES), Grant No. 00.889.834/0001-08.

ORCID and License

References

[1] Amir A, Paryenty H, and Roditty L. Configurations and minority in the string consensus problem. Algorithmica. 2016; 74(4):1267-1292.
[2] Basavaraju M, Panolan F, Rai A, Ramanujan MS, Saurabh S. On the kernelization complexity of string problems. Theoretical Comp. Science. 2018; 730:21–31.
[3] Boucher C, Brown DG, and Durocher S. On the Structure of Small Motif Recognition Instances.. Berlin Heidelberg: Springer; 2009. p.269-281.
[4] Fomin FV, Lokshtanov D, Saurabh S, Zehavi M. Kernelization: Theory of Parameterized Preprocessing. Great Britain: Cambridge University Press; 2019. chapter 1.
[5] Frances M and Litman A. On covering problems of codes. Theor. Comput. Syst. 1997; 30:113-119.
[6] Gramm J, Niedermeier R, Rossmanith P. Exact solutions for closest string and related problems. In: ISAAC '01. Proceedings of the 12th International Symposium on Algorithms and Computation; 2001. p.441-453.
[7] Gramm J, Niedermeier R, Rossmanith P. Fixed-parameter algorithms for closest string and related problems. Algorithmica. 2003; 37(1):25-42.
[8] Lenstra Jr HW. Integer programming with a fixed number of variables. Mathematics of Operations Research. 1983; 8(4):538-548.
[9] Liu X, Liu S, Hao Z, Mauch H. Exact algorithm and heuristic for the closest string problem. Computers & Operations Research. 1011; 38(11):1513-1520.

[10] Ma B, Sun X. More Efficient Algorithms for Closest String and Substring Problems. Berlin Heidelberg; Springer; 2008. p. 396-409.

[11] Storer JA. Data compression: Methods and theory. New York: Computer Science Press, Inc. 1988. chapter 1.

[12] Vilca OL. *Combinatorial Approaches for the Closest String Problem.*[Doctoral thesis]. Amazonas: Federal university of Amazonas, Computing institute. 2019. Recovered from https://tede.ufam.edu.br/handle/tede/7449.