Optimal Solutions for the Closest String Problem via Integer Programming

Cláudio N. Meneses [*] [‡] Zhaosong Lu [†]
Carlos A.S. Oliveira [‡] Panos M. Pardalos [‡]

Abstract

In this paper we study the Closest String Problem (CSP), which can be defined as follows: given a finite set $\mathcal{S} = \{s^1, s^2, \ldots, s^n\}$ of strings, each string with length m, find a median string t of length m minimizing d, such that for every string $s^i \in \mathcal{S}$, $d_H(t, s^i) \leq d$. By $d_H(t, s^i)$ we mean the Hamming distance between t and s^i. This is a NP-hard problem, with applications in Molecular Biology and Coding Theory. Even though there are good approximation algorithms for this problem, and exact algorithms for instances with d constant, there are no studies trying to solve it exactly for the general case. In this paper we propose three integer programming formulations and a heuristic, which is used to provide upper bounds on the value of an optimal solution. We report computational results of a branch-and-bound algorithm based on one of the IP formulations, and of the heuristic, executed over randomly generated instances. These results show that it is possible to solve CSP instances of moderate size to optimality.

Keywords: *Computational Biology; Mathematical Programming; Branch-and-Bound Algorithms; Closest String Problem.*

1. Introduction

In many problems occurring in computational biology, there is the need of comparing, and finding common features in sequences. In this paper we study one such problem, known as the *Closest String Problem* (CSP). The CSP has important applications not only in computational biology, but also in other areas, such as coding theory.

In the CSP, the objective is to find a string t which minimizes the number of differences among elements in a given set $\mathcal{S}$ of strings. The distance here

[*]This author is supported by the Brazilian Federal Agency for Post-Graduate Education (CAPES) - Grant No. 1797-99-9.

[†]School of Industrial & Systems Engineering, Georgia Institute of Technology, email: zhaosong@isye.gatech.edu

[‡]Dept. of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL, 32611, USA. Email addresses: {claudio,oliveira,pardalos}@ufl.edu

is defined as the Hamming distance between t and each of the $s^i \in \mathcal{S}$. The Hamming distance between two strings a and b of equal length is calculated by simply counting the character positions in which a and b differ.

The CSP has been widely studied in the last few years [4, 6, 7, 12, 13, 15], due to the importance of its applications. In Molecular Biology problems, the objective is to identify strings that correspond to sequences of a small number of chemical structures. A typical case where the CSP is useful occurs when, for example, it is needed to design genetic drugs with structure similar to a set of existing sequences of RNA [7]. As another example, in Coding Theory [14] the objective is to find sequences of characters which are closest to some given set of strings. This is useful in determining the best way of encoding a set of messages.

The CSP is known to be NP-hard [3]. Even though there are good approximation algorithms for this problem (including a PTAS [10]), and exact algorithms for instances with constant difference [1], there are no studies reporting on the quality of integer programming formulations for the general case, when the maximum distance d between a solution and the strings in $\mathcal{S}$ is variable.

Problem Definition We give here a formal definition of the CSP. To do this, we first introduce some notation. An alphabet $\mathcal{A} = \{c_1, \ldots, c_k\}$ is a finite set of elements, called characters, from which strings can be constructed. Each string s is a sequence of characters $(s_i, \ldots, s_m)$, $s_i \in \mathcal{A}$, where $\mathcal{A}$ is the alphabet used. The size of a string $\alpha(s)$ is the number of elements in the character sequence that composes the string. Thus, if $s = \{s_1, \ldots, s_m\}$, then $\alpha(s) = m$.

We define the *Hamming distance* $d_H(a, b)$, between two strings a and b, such that $\alpha(a) = \alpha(b)$, as the number of positions in which they differ. Let us define the predicate function $\mathcal{E} : \mathcal{A} \times \mathcal{A} \rightarrow \{0, 1\}$ as $\mathcal{E}(x, y) = 1$ if and only if $x \neq y$. Thus, we have $d_H(a, b) = \sum_{i=1}^{\alpha(a)} \mathcal{E}(a_i, b_i)$. For instance,

$$d_H(\text{“}\texttt{CATC11058}\text{”}, \text{“}\texttt{CAA18970T}\text{”}) = 7.$$

Let $\mathcal{A}^k$ be the set of all strings s over the alphabet $\mathcal{A}$ with length $\alpha(s) = k$. Then, the CSP is defined as follows: given a finite set $\mathcal{S} = \{s^1, s^2, \ldots, s^n\}$ of strings, with $s^i \in \mathcal{A}^m$, $1 \leq i \leq n$, find a string $t \in \mathcal{A}^m$ minimizing d such that, for every string $s^i \in \mathcal{S}$, $d_H(t, s^i) \leq d$.

The following example shows an instance of the CSP and its optimal solution.

Example 1 *Suppose $\mathcal{S} = \{$ “differ”, “median”, “length”, “medium”$\}$. Then, $t =$ “menfar” constitutes an optimal solution to $\mathcal{S}$, and the corresponding minimum difference is $d = 4$.*

Previous Work The recent developments of Computational Biology applications has required the study of optimization problems over sequences of characters. Some early developments have been done using statistical methods [6]. Li *et al.* [9] initially explored the combinatorial nature of the CSP, and proved that

it is NP-hard. Li *et al.* also described some applications in Molecular Biology, where this problem has been very useful.

In [5], Gramm *et al.* have shown that for a fixed value of d it is possible to solve the problem in polynomial time. This can be done using techniques of fixed parameter computation.

Approximation algorithms have also being used to give near optimal solutions for the CSP. In [7], for example, an algorithm with performance guarantee of $\frac{4}{3}(1+\epsilon)$, for any small constant $\epsilon > 0$, is presented and analyzed, with a similar result appearing also in [4]. A PTAS for the CSP was presented by Li *et al.* [10].

Contributions In this paper we are interested in optimal solutions for the general case of the CSP, where the maximum distance between the median string and the set of strings is variable. With this objective, we propose three integer programming formulations and explore properties of these formulations. We also show computational results of a branch-and-bound algorithm that uses the linear relaxation of one of the proposed formulations.

In order to speed up the branch-and-bound algorithm, we designed and implemented a heuristic for finding upper bounds on the value of optimal solutions for the CSP. The computational experiments show that the resulting approach is effective in solving CSP instances of moderate size to optimality.

Outline This paper is organized is follows. In Section 2 we present three Integer Programming (IP) formulations for the CSP. The formulations are compared and we show that the third formulation is the strongest.

In Section 3, we describe the branch-and-bound algorithm based on one of the formulations presented. We discuss several parameters used in this implementation, and how they were tuned to improve the performance of the algorithm. Furthermore in this section, we present a new heuristic algorithm for the CSP. In Section 4 we show computational results obtained by the proposed methods. Finally, concluding remarks are given in section 5.

2. Integer Programming Formulations

In this section we present three IP formulations, and show that these formulations correctly describe the CSP. Our main result in this section states that the third formulation is the strongest of them. In Subsection 2.1 we introduce definitions and notations. The IP formulations are given in Subsection 2.2, 2.3 and 2.4.

2.1 Definitions and Notation

An *instance* of the CSP consists of a finite set $\mathcal{S} = \{s^1, s^2, \ldots, s^n\}$, such that $\alpha(s^i) = m$, for $1 \leq i \leq n$. However, instead of working directly on strings $s^i = (s^i_1, s^i_2, \ldots, s^i_m) \in \mathcal{S}$, for $1 \leq i \leq n$, one can transform them into points

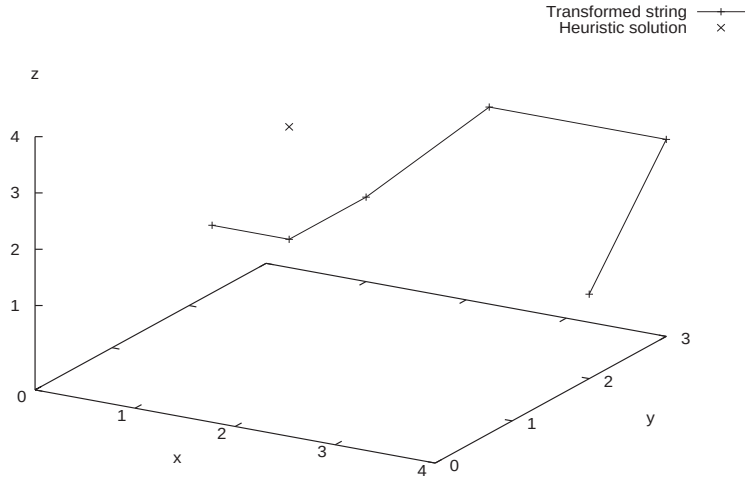

Figure 1: Example of transformation from strings to points in $\mathbf{Z}^3$.

$x^i \in \mathbf{Z}^m$. That is, a unique natural number can be assigned to each character s^i_k, for $k = 1, \ldots, m$. To formalize this, let $\pi : \mathcal{A} \to \mathbf{Z}$ be an assignment of characters to integers, such that for $a, b \in \mathcal{A}$, if $a \neq b$, then $\pi(a) \neq \pi(b)$. In this paper, the assignment $\pi(a) = 1, \pi(b) = 2, \ldots, \pi(z) = 26$ is used, unless otherwise stated. Now, let $\phi_\pi \colon \mathcal{A}^m \to \mathbf{Z}^m$ be a function mapping characters to points in $\mathbf{Z}^m$. We define $\phi_\pi(s) = x$, such that $x_k = \pi(s_k)$, for $k \in \{1, \ldots, m\}$.

For instance, in Example 1 we have $\phi_\pi(x^1) = (4, 9, 6, 6, 5, 18)$, $\phi_\pi(x^2) = (13, 5, 4, 9, 1, 14)$, $\phi_\pi(x^3) = (12, 5, 14, 7, 20, 8)$, and $\phi_\pi(x^4) = (4, 9, 6, 6, 21, 13)$. We extend the definition of the Hamming function d_H to sequences of integers, in a similar way as to strings of characters in $\mathcal{A}$, and note that $d_H(s^i, s^j) = k$ if and only if $d_H(x^i, x^j) = k$. Let $\mathcal{S}_\mathcal{T}$ denote the set of points $x^i \in \mathbf{Z}^m$ obtained from an instance $\mathcal{S}$ of the CSP by application of the transformation ϕ_π. We refer to $\mathcal{S}_\mathcal{T}$ and $\mathcal{S}$ interchangeably as an instance of the CSP, to simplify notation in the remaining of the paper.

As a second example, consider $\mathcal{S} = \{AAT, ATA, AGA, GTC, CGG, CTA\}$. Using the assignment function $\pi' \colon \mathcal{A} \to \mathbf{Z}$ defined as $\pi'(A) = 1$, $\pi'(T) = 2$, $\pi'(G) = 3$, and $\pi'(C) = 4$, the transformed instance is $\mathcal{S}_\mathcal{T} = \{(1, 1, 2)$,$(1, 2, 1)$, $(1, 3, 1)$, $(3, 2, 4)$, $(4, 3, 3)$, $(4, 2, 1)\}$. The points in $\mathcal{S}_\mathcal{T}$ are plotted in Figure 1, where they appear joined by a line. The solution given by a heuristic algorithm (to be presented later) is the point $s = (1, 2, 3)$.

Let $G = (V, A)$ be a directed graph. Then, a *directed path* is a sequence of distinct nodes such that for each adjacent pair of nodes v_i and v_{i+1}, there is an arc $(v_i, v_{i+1}) \in A$. By $|V|$ we denote the cardinality of set V.

2.2 First IP Formulation

We present initially a simple IP formulation, which will be later improved, according to some properties of optimal solutions of the CSP. Define the variables $t_k \in \mathbf{Z}$ for $k = 1, \ldots, m$ and the variables

$$z_k^i = \begin{cases} 1 & \text{if } t_k \neq x_k^i \\ 0 & \text{otherwise.} \end{cases}$$

Then, we have the following IP formulation.

$$\begin{aligned}
\text{P1}: \quad & \min d && && (1)\\
\text{s.t.} \quad & \sum_{k=1}^{m} z_k^i \leq d && i = 1, \ldots, n && (2)\\
& t_k - x_k^i \leq K z_k^i && i = 1, \ldots, n;\ k = 1, \ldots, m && (3)\\
& x_k^i - t_k \leq K z_k^i && i = 1, \ldots, n;\ k = 1, \ldots, m && (4)\\
& z_k^i \in \{0, 1\} && i = 1, \ldots, n;\ k = 1, \ldots, m && (5)\\
& d \in \mathbf{Z}_+, && && (6)\\
& t_k \in \mathbf{Z} && k = 1, \ldots, m && (7)
\end{aligned}$$

where $K = |\mathcal{A}|$ is the size of the alphabet used by instance $\mathcal{S}$. In this formulation, d represents the maximum difference between t and the strings in $\mathcal{S}$, and is the value to be minimized. Constraints (2) give a lower bound for the value of d, since it must be greater than or equal to the total number of differences, for each string $s^i \in \mathcal{S}$. Taken together, constraints (3) and (4) give lower and upper bounds to the difference $t_k - x_k^i$, and it is therefore equivalent to $|t_k - x_k^i| \leq K$, whenever $z_k^i = 1$, and $t_k = x_k^i$, when $z_k^i = 0$. Constraints (5)-(7) give the integrality requirements.

In the next theorem, we prove that P1 is a correct formulation for the CSP.

Theorem 1 *A vector $t \in \mathbf{Z}^m$ defines a feasible solution to an instance $\mathcal{S}_\mathcal{T}$ of the CSP if and only if t satisfies the constraints in P1.*

Proof: Let $\mathcal{S}_\mathcal{T}$ be an instance of the CSP. It is clear that any vector in $\mathbf{Z}^m$ defines a feasible solution to $\mathcal{S}_\mathcal{T}$. Therefore, a solution to P1 is also a solution to $\mathcal{S}_\mathcal{T}$.

If t defines a feasible solution to $\mathcal{S}_\mathcal{T}$, then assign values to z_k^i, for $i = 1, \ldots, n$, $k = 1, \ldots, m$, as follows:

$$z_k^i = \begin{cases} 1 & \text{if } t_k \neq x_k^i \\ 0 & \text{otherwise.} \end{cases}$$

Now, define

$$d = \max_{1 \leq i \leq n} \sum_{k=1}^{m} z_k^i.$$

Taking d and z_k^i as above, all constraints (2) to (6) will be satisfied. Thus, P1 is a correct formulation of the CSP. □

Now, we present an observation that will be used to improve the previous formulation for the CSP.

Proposition 2 *Let $\mathcal{S}$ be an instance of the CSP. Then, there is an optimal solution t of $\mathcal{S}$, such that t_k is the character in position k in at least one of the strings of $\mathcal{S}$.*

Proof: Let $z(s)$ be the objective cost of solution s, for a given instance of the CSP. Suppose that we are given an optimal solution t, such that t_k does not appear in position k on any of the strings $s^i \in \mathcal{S}$, for $i \in \{1, \ldots, n\}$. Consider now the string t' defined in the following way:

$$t' = \begin{cases} t_i & \text{if } i \neq k \\ s_i^1 & \text{if } i = k. \end{cases}$$

We claim that t' gives a solution not worse than t. This happens because for each string s^i, with $i \neq 1$, the value of $d_H(t', s^i) = d_H(t, s^i)$. Also, according to the definition of t', $d_H(t', s^1) = d_H(t, s^1) - 1$. Thus, $z(t') \leq z(t)$.

Now, given any optimal solution t, if there is a set $Q \subseteq \{1, \ldots, n\}$ such that t_j, for $j \in Q$, does not appear in position j in any of the strings $s^i \in \mathcal{S}$, then we can do the following procedure. Let $t^0 = t$. Then, for each $i \in \{1, \ldots, l = |Q|\}$, let k be the ith element of Q, and construct a solution t^i from t^{i-1} using the method above. Then, as t is optimal, $z(t^l) = z(t)$ and t^l has the desired property. □

Given an instance $\mathcal{S}_\mathcal{T}$ of the CSP, let Ω be the bounded region defined by the points $x^i \in \mathcal{S}_\mathcal{T}$, and Ω_k the smallest region of $\mathbf{Z}$ such that $x_k^i \in \Omega_k$, for each $s^i \in \mathcal{S}$. Define the diameter of Ω_k, for each $k \in \{1, \ldots, m\}$, as

$$\text{diam}(\Omega_k) = \max\{|x_k^i - x_k^j| \colon x^i, x^j \in \mathcal{S}_\mathcal{T}, 1 \leq k \leq m\},$$

with $|\cdot|$ standing for the absolute value function. For instance, in Example 1 $\text{diam}(\Omega_1) = 9$, $\text{diam}(\Omega_2) = 4$, and $\text{diam}(\Omega_3) = 10$.

Using the definition above, an easy consequence of Proposition 2 is the following.

Proposition 3 *For each instance $\mathcal{S}$ of the CSP, let x^i be the point in $\mathbf{Z}^m$ corresponding to s^i. Then, there is an optimal solution $x \in \mathbf{Z}^m$ such that for each $s^i \in \mathcal{S}$, $x_k^i = x_k$ or $x_k^i - x_k \leq$ diam(Ω_k).*

Proof: Given an optimal solution t, apply Proposition 2. Then, we find another optimal solution t' such that each character t'_k appears in position k of some string $s^i \in \mathcal{S}$. Applying on t' the transformation ϕ_π from $\mathcal{S}$ to $\mathbf{Z}^m$, we find the point x with the desired property. □

We can, therefore, use Proposition 3 to improve formulation P1. The constraints (3) and (4) can be changed to the following:

$$t_k - x_k^i \leq \text{diam}(\Omega_k) z_k^i \qquad i = 1, \ldots, n;\ k = 1, \ldots, m \qquad (8)$$

$$x_k^i - t_k \leq \text{diam}(\Omega_k) z_k^i \qquad i = 1, \ldots, n;\ k = 1, \ldots, m. \qquad (9)$$

It is important to notice that the resulting formulation, which we will call P1A, does not include all feasible solutions to the CSP. For example, if $\mathcal{S} = \{$*"ABC"*, *"DEF"*$\}$, the optimal solution has $d = 2$. An optimal string is $t =$ *"EBF"*, which has distance $d = 2$, although it is not feasible for P1A. However, according to Preposition 2, any optimal solution to P1 has a corresponding solution in P1A. Thus, to find an optimal solution to any instance of the CSP, it suffices to solve problem P1A.

The resulting formulation P1A is therefore an strengthening of P1, and has $n + 3nm + 1$ constraints and $m + nm + 1$ variables.

2.3 Second IP Formulation

In this subsection we provide an IP formulation that has a nice interpretation as a directed graph. Recall Proposition 2. Using this proposition, we can describe the process of finding a solution in the following way. Set $i = 1$. Then, select one of the characters $c \in \{s_i^1, \ldots, s_i^n\}$ to be in the ith position of t. Repeat this while $i \leq m$, and return the resulting string t.

Alternatively, this process can be thought as finding a path in a directed graph. Let $G = (V, A)$ be a directed graph with $V = V_1 \cup V_2 \cup \cdots \cup V_m \cup \{F, D\}$, where V_j is the set of characters used in the jth position of the $s^i \in \mathcal{S}$, i.e., $V_j = \bigcup_{k=1}^n s_j^k$. There is an arc between each pair of nodes v, u such that $v \in V_i$ and $u \in V_{i+1}$, for $i \in \{1, \ldots, m-1\}$. There is also an arc between F and each node in V_1, and between each node in V_m and D. For example, the graph corresponding to the strings in Example 1 is shown in Figure 2.

Given an instance $\mathcal{S}_\mathcal{T}$ of CSP, a feasible solution can be easily identified by creating G as described above and finding a directed path from node F to node D. For instance, taking the directed path $(F, 4, 9, 6, 6, 1, 13, D)$ in the graph shown in Figure 2, and discarding nodes F and D, we obtain the feasible solution $(4, 9, 6, 6, 1, 13)$ which corresponds to string *"diffam"*.

One can think of enumerating all directed paths from F to D and choose that one with minimum maximum Hamming distance to all strings in $\mathcal{S}_\mathcal{T}$. However, the number of such paths is given by $\prod_{k=1}^m |V_k|$. Since $|V_k|$ can be equal to n in the worst case, it follows that $\prod_{k=1}^m |V_k|$ can be equal to n^m. For small values of n and m one could try that approach, but for large values this becomes impractical. However, we can try to use this idea to strengthen formulation P1A.

Define the variables

$$v_{j,k} = \begin{cases} 1 & \text{if node } j, \text{ for } j \in V_k, \text{ is used in a solution} \\ 0 & \text{otherwise.} \end{cases}$$

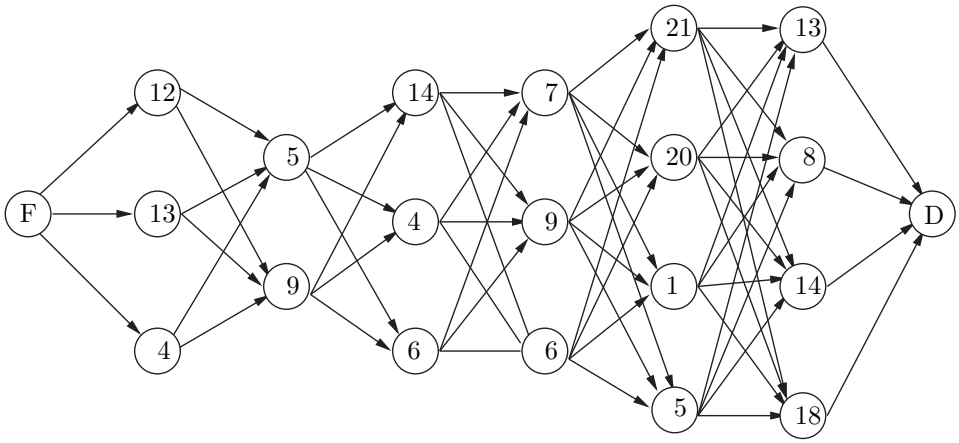

Figure 2: Directed graph $G = (V, A)$ for strings in Example 1.

Then, another IP formulation is defined in the following way

$$\text{P2}: \quad \min d \tag{10}$$

$$\text{s.t.} \quad \sum_{j \in V_k} v_{j,k} = 1 \qquad k = 1, \ldots, m \tag{11}$$

$$\sum_{j \in V_k} j v_{j,k} = t_k \qquad k = 1, \ldots, m \tag{12}$$

$$\sum_{k=1}^{m} z_k^i \leq d \qquad i = 1, \ldots, n \tag{13}$$

$$t_k - x_k^i \leq \text{diam}(\Omega_k) z_k^i \qquad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{14}$$

$$t_k - x_k^i \leq \text{diam}(\Omega_k) z_k^i \qquad i = 1, \ldots, n;\ k = 1, \ldots, m. \tag{15}$$

$$v_{j,k} \in \{0, 1\} \qquad j \in V_k;\ k = 1, \ldots, m \tag{16}$$

$$z_k^i \in \{0, 1\} \qquad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{17}$$

$$d \in \mathbf{Z}_+. \tag{18}$$

$$t_k \in \mathbf{Z} \qquad k = 1, \ldots, m \tag{19}$$

The additional inequalities (11) ensure that only one node in each V_k is chosen. Moreover, constraints (12) force t_k to be one of the elements in V_k. Constraints (16) enforce integrality for variables $v_{j,k}$. The remaining constraints are equivalent to P1A. In the next theorem we prove that P2 is a correct formulation for the CSP.

Theorem 4 *Given an instance $\mathcal{S}$ of CSP, solving the corresponding formulation P2 is equivalent to finding an optimal solution for $\mathcal{S}$.*

Proof: Clearly, any solution x for P2 is also feasible for P1. Therefore, by Theorem 1, x is a solution to $\mathcal{S}$. Now, recalling Proposition 2, for any optimal solution to P1 we can find a solution x' with the same objective cost and with x'_k appearing in position k in at least one of the strings $s^i \in \mathcal{S}$. Then, x' satisfies constraints (11) and (12). This implies that the optimal solution of P2 has the same objective value as the optimal solution of P1, which is also an optimal solution for instance $\mathcal{S}$. □

Formulation P2 is interesting as a way of reducing the size of the feasible solution space in formulation P1A. For example, let $\mathcal{S} = \{$*"ABC", "DEF"*$\}$. Then, *"BBF"* is a feasible solution for P1A, but not for P2, and therefore P2 $\subset$ P1A. However, from the point of view of the resulting linear relaxation, P1A gives the same result as P2, as shown in the next theorem.

Theorem 5 *Let RP1A and RP2 be the continuous relaxations of formulations P1A and P2, respectively. If z_1^* is the optimal value of RP1A and z_2^* is the optimal value of RP2, then $z_1^* = z_2^*$.*

Proof: We know that P2 $\subset$ P1A, which implies RP2 $\subseteq$ RP1A. Thus, $z_1^* \leq z_2^*$. Then, it suffices to prove that $z_2^* \leq z_1^*$. Suppose we are given a solution t to RP1A with cost z_1^*. We claim that t also satisfies RP2, and therefore t is a solution for RP2 with the same objective cost. This is easy to establish for constraints (13)-(15). Note however that constraints (11) and (12) together state that t_k is a linear combination of the values $j \in V_k$, for $k \in \{1, \ldots, m\}$. An easy way to satisfy these equations is the following. Solve

$$t_k = \lambda a + (1 - \lambda) b$$

for λ, where a and b are, respectively, the smallest and the greatest element in V_k. Then, make the assignments $v_{a,k} = \lambda$, $v_{b,k} = (1 - \lambda)$, and $v_{j,k} = 0$, for $j \in V_k \setminus \{a, b\}$. □

We note that the number of variables and constraints in P2 are given by $m + nm + m|V_k| + 1$ and $3nm + 2m + n + m|V_k| + 1$.

2.4 Third IP Formulation

In this section we propose another IP formulation for the CSP using the idea in Proposition 2. We use the same definition of variables $v_{j,k}$ given in Subsection 2.3.

The third IP formulation is given by

$$\text{P3}: \quad \min\ d \tag{20}$$

$$\text{s.t.} \quad \sum_{j \in V_k} v_{j,k} = 1 \qquad k = 1, \ldots, m \tag{21}$$

$$m - \sum_{j=1}^{m} v_{s^i_j, j} \leq d \qquad i = 1, \ldots, n \tag{22}$$

$$v_{j,k} \in \{0, 1\} \qquad j \in V_k,\ k = 1, \ldots, m \tag{23}$$

$$d \in \mathbf{Z}_+. \tag{24}$$

Inequalities (21) guarantee that only one node in each V_k is selected. Inequalities (22) say that if a node in a string s^i is not in a solution t, then that node will contribute to increase the Hamming distance from t to s^i. Constraints (23) are binary inequalities, and (24) forces d to assume a nonnegative integer value.

We now show that this is a correct formulation.

Theorem 6 *Given an instance $\mathcal{S}$ of CSP, solving the corresponding formulation P3 is equivalent to finding an optimal solution for $\mathcal{S}$.*

Proof: Let $\mathcal{S}$ be an instance of the CSP. It is clear that a solution to P3 is also feasible for $\mathcal{S}$, since any $x \in \mathbf{Z}^m$ is feasible for $\mathcal{S}$. Now, if t defines a feasible solution to $\mathcal{S}$ satisfying Proposition 2, then $P = (F, t_1, \ldots, t_m, D)$ defines a directed path from F to D in the graph G, constructed from $\mathcal{S}$ as described in the previous section. Assign values to the variables $v_{j,k}$ as follows

$$v_{i,k} = \begin{cases} 1 & \text{if } i = t_k \\ 0 & \text{otherwise.} \end{cases}$$

This definition ensures that constraints (21) are satisfied. Notice also that in constraint (22) the value of $\sum_{j=1}^{m} v_{s^i_j, j}$ is at most m. Hence, $d \geq 0$, in accordance with constraint (24). Constraints (23) are satisfied as well.

Now, using Proposition 2, any instance $\mathcal{S}$ of the CSP has at least one optimal solution that satisfies that property. Thus, the optimal solution of P3 has the same value of an optimal solution of $\mathcal{S}$. □

We note that the number of variables and constraints in P3 are $m|V_k| + 1$ and $m + n + m|V_k| + 1$ respectively. In the next theorem, we determine the relationship between P3 and the previous formulations.

Theorem 7 *Let z_1^* and z_3^* be the optimal values of linear relaxations of formulations P1 and P3, respectively. Then $z_1^* \leq z_3^*$.*

Proof: Let RP1 and RP3 be the linear relaxations of P1 and P3 respectively. To show that $z_1^* \leq z_3^*$ it is enough to show that, given any vector y satisfying the constraints of RP3, y can be transformed into a vector x satisfying the constraints of RP1. Given this transformation, the value of x must be at least z_1^*, since z_1^* was assumed to be the optimal value of the linear relaxation of P1.

We shall show that any feasible solution for RP3 is also feasible for RP1. If d, $v_{j,k}$, for $j \in V_k$, $k = 1, \ldots, m$, is a feasible solution for RP3, we can define a feasible solution d, t_k, z_k^i, where $i \in \{1, \ldots, n\}$, $k \in \{1, \ldots, m\}$, for RP1 in the following way. Set

$$\begin{aligned} t_k &= \sum_{j \in V_k} j v_{j,k} \qquad \text{for } k \in \{1, \ldots, m\}, \qquad \text{and} \\ z_k^i &= 1 - v_{s_k^i,k} \qquad \text{for } k \in \{1, \ldots, m\},\ i = \{1, \ldots, n\}. \end{aligned}$$

Constraints (5) are automatically satisfied by this definition. From the second constraint of RP3, we have

$$d \geq m - \sum_{k=1}^{m} v_{s_k^i,k} = \sum_{k=1}^{m} (1 - v_{s_k^i,k}) = \sum_{k=1}^{m} z_k^i.$$

So, Constraints (2) and (6) are also satisfied. It remains to show that Constraints (3) and (4) are satisfied as well. Clearly we have

$$t_k - x_k^i = \sum_{j \in V_k} j v_{j,k} - x_k^i = \sum_{j \in (V_k \setminus \{x_k^i\})} j v_{j,k} + x_k^i v_{x_k^i,k} - x_k^i$$

Now, if $t_k - x_k^i \geq 0$,

$$\begin{aligned} |t_k - x_k^i| &= t_k - x_k^i = \sum_{j \in V_k \setminus \{x_k^i\}} j v_{j,k} - x_k^i (1 - v_{x_k^i,k}) \\ &\leq \sum_{j \in V_k \setminus \{x_k^i\}} j v_{j,k} \leq K \sum_{j \in V_k \setminus \{x_k^i\}} v_{j,k} \\ &= K(1 - v_{x_k^i,k}) \\ &= K z_k^i, \end{aligned}$$

where $K = |\mathcal{A}|$ is the size of the alphabet used by instance $\mathcal{S}$. Similarly, if $t_k - x_k^i \geq 0$, we have

$$\begin{aligned} |t_k - x_k^i| &= x_k^i - t_k = x_k^i (1 - v_{x_k^i,k}) - \sum_{j \in V_k \setminus \{x_k^i\}} j v_{j,k} \\ &\leq x_k^i (1 - v_{x_k^i,k}) \\ &\leq K(1 - v_{x_k^i,k}) \\ &= K z_k^i. \end{aligned}$$

Hence, in both cases Constraints (3) and (4) in RP1 are satisfied by the solution defined above. □

Note that from the point of view of the feasible set, P3 is similar to P2, since P3 ⊂ P1 and P2 ⊂ P1A ⊂ P1. However, the relaxation RP3 of P3 has shown to be better than the relaxation RP2 of P2. Recalling Theorem 5, RP2 always gives the same value as RP1A. On the other hand, RP3 gives in practice results much better than RP1A, as demonstrated in the computational results section.

3. Implementation Issues

Using the formulations described in the previous section, we developed a Branch-and-Bound algorithm (B&B) to solve the CSP. We describe in this section the parameters used in the B&B, as well as the methodology used to create upper bounds for instances of the problem. Then, we discuss a heuristic for the CSP, which was designed to find good initial solutions. These solutions can be used as an upper bound to speed up the B&B operation.

3.1 Branch-and-Bound Algorithm

Branch-and-bound algorithms are a very important tool in the practical solution of integer programming problems. Much work has been done in the development of B&B, and an introduction to the technique is given by Lee and Mitchell [8]. In this paper a B&B is proposed using one of the formulations shown in the previous section. A simple description of the B&B algorithm for minimization problems is the following.

1. *Initialization.* Start with an IP formulation. Initialize the B&B search tree to one node with this formulation, and select this node for the next phase. Let s be a feasible solution to the CSP and let UB be the objective function value associated with s.

2. *Subproblem solving.* If the search tree is empty, then stop the execution. Otherwise, take the selected node and solve the LP relaxation using a LP solver. If the solution is fractional and greater than UB, or if the LP is infeasible, leave this node and go to step 5. If all variables are integer and they define a feasible solution, this is a new upper bound that can be used to update UB. Else, run a primal heuristic in order to construct a feasible solution from the existing LP information. If the resulting solution has cost less than UB, update UB using this value.

3. *Pruning.* If UB was decreased in the previous step, remove all nodes in the search tree with objective cost greater than UB.

4. *Branching phase.* Select one of the fractional variables in the current solution of the LP relaxation and create two child nodes, were the selected variable is set to 0 and 1, respectively.

5. *Node selection.* Select a non-explored node in the search tree, and go back to step 2.

The general description of the B&B procedure allows many decisions for the algorithm designer. The first important decision concerns the correct IP formulation for the problem. Using the results in Theorems 5 and 7, and with some computational experimentation, we found that formulation P3 is the most interesting from the computational point of view (see Table 1). The computational effort to solve the CSP with P3 is smaller since there are less constraints

and variables, and the formulation cuts off many solutions that are not needed to find the optimal. Taking this fact into consideration, the relaxation of P3 was used on the B&B procedure.

In Step 2 of the previous algorithm, the method used to generate an integer solution from a fractional solution is called a *primal heuristic*. An efficient primal heuristic is important for the success of a B&B implementation, since it allows the upper bound to be reduced in a way that explores the underlying structure of the problem. In the B&B implementation, the heuristic shown in Algorithm 1 is used as a primal heuristic. The solution is found by selecting the character that has highest value on the optimal fractional solution. This is done for each of the m positions, until a feasible solution is found. Clearly, the computational complexity of Algorithm 1 is $O(nm)$.

Algorithm 1: Primal heuristic for formulation P3.

Input: Fractional solution v, d
Output: Feasible solution v, d
for $k \leftarrow 1$ **to** m **do**
  $max \leftarrow -1$; $i \leftarrow 1$
  for $j \in V_k$ **do**
    if $(v_{j,k} > max)$ **then**
      $max \leftarrow v_{j,k}$
      $i \leftarrow j$
    end
  end
  for $j \in V_k$ **do** $v_{j,k} \leftarrow 0$
  $v_{i,k} \leftarrow 1$
end
$d \leftarrow \max_{i \in \{1,\ldots,n\}}(m - \sum_{j=1}^{m} v_{s_j^i,j})$

The branching phase of the B&B requires a policy for selection of a variable or constraint where the current node will be branched. In the B&B for the CSP, only branching on variables is employed. The criterion for variable selection is based on the value of the variables in a fractional solution. Given an optimal fractional solution x' for the linear relaxation of P3, we branch on the fractional variable x_j with *maximum* value of x'_j.

Finally, the node selection is another important policy that must be defined in the B&B. In the algorithm used, the next node to be explored in the enumeration tree is the one with smallest linear relaxation value (also known as *best bound*).

3.2 Generation of Upper Bounds

An important performance consideration in a B&B algorithm is the initial upper bound used. A good initial upper bound will improve the running time, since the number of nodes that need to be explored can be reduced as a result of pruning.

With this objective, we propose a heuristic for the CSP, shown in Algorithm 2. The heuristic consists of taking one of the strings in $\mathcal{S}$ and modifying it until a new locally optimal solution is found.

In the fist step, the algorithm searches for a solution $s \in \mathcal{S}$ which is closest to all other strings in $\mathcal{S}$. This step is explained in Algorithm 3. In the second step, the distance d between s and the remaining strings is computed. In the last step of Algorithm 2, a local search procedure is applied as follows. Let r be the string in $\mathcal{S}$ such that $d_H(r, s^i)$, where $i \in \{1, \ldots, n\}$ and $r \neq s^i$, is maximum, and let s be the current solution. Then, for $i \in \{1, \ldots, m\}$, if $s_i \neq r_i$ and replacing s_i by r_i does not make the solution s worse, the replacement is done and the Hamming distances from s to all strings in $\mathcal{S}$ are updated. After we have scanned all of m positions, a new string r is selected among the strings in $\mathcal{S}$ that is farthest from the resulting s, and the process is repeated. The number of repetitions is controlled by the parameter N. The details of the local search step are presented in Algorithm 4.

Algorithm 2: Generate upper bound for the CSP.

Input: instance $\mathcal{S}$, N, *seed*

Output: string s, distance d

1 $s \leftarrow$ string in $\mathcal{S}$ closest to the other $s^i \in \mathcal{S}$
2 $d \leftarrow \max_{i \in \{1,\ldots,n\}} d_H(s, s^i)$
3 improve_solution(s, d, N)

We now analyze the computational complexity of Algorithm 2.

Theorem 8 *Algorithm 2 has time complexity $O(nmN)$.*

Proof: The computation of Hamming distance can be done in $O(m)$. It follows that Step 1 (Algorithm 3) has time complexity $O(mn^2)$. Step 2 consists of n Hamming distance computations, and can be clearly implemented in $O(nm)$. Finally, Algorithm 4 takes $O(nmN)$. Hence, the total time complexity of Algorithm 2 is $O(nmN)$, for $N \geq n$. □

Algorithm 3: Step 1 of Algorithm 2.

Input: instance $\mathcal{S}$, current solution s and distance d, N
Output: resulting solution s and distance d
$min \leftarrow m + 1$
for $i \leftarrow 1$ **to** n **do**
  $max \leftarrow -1$
  for $j \leftarrow 1$ **to** n such that $j \neq i$ **do**
    if $(max < d_H(s^i, s^j)$ **then**
      $max \leftarrow d_H(s^i, s^j)$
    end
  end
  if $(max \neq -1)$ **and** $(max < min)$ **then**
    $s \leftarrow s^i$; $d \leftarrow max$
    $min \leftarrow max$
  end
end

4. Computational Experiments

We now present the computational experiments carried out with the algorithms described in the previous section. Initially, we describe the set of instances used in the tests. Then, in Subsection 4.2 the results obtained by the branch-and-bound algorithm and by the proposed heuristic are discussed.

4.1 Instances and Test Environment

Two classes of instances were tested. In the first class, the alphabet $\mathcal{A}$ is the set $\{0, 1\}$, representing binary strings. This kind of instances is interesting when the strings compared represent digital information, such as for example in computer generated data. The second class of instances uses the alphabet $\mathcal{A} = \{A, T, G, C\}$. This is the basic alphabet used in biological applications, where the characters represent proteins in the DNA of RNA structure.

The instances used were generated randomly, in the following way. Given parameters n (number of strings), m (string length) and an alphabet $\mathcal{A}$, randomly choose a character from $\mathcal{A}$ for each position in the resulting string.

The algorithm used for random number generation is an implementation of the multiplicative linear congruential generator [11], with parameters 16807 (multiplier) and $2^{31} - 1$ (prime number).

All tests were executed on a Pentium 4 CPU with speed of 2.80GHz and 512MB of RAM, under WindowsXP. The heuristic algorithm was implemented in C++ language, and the XPress Mosel software [2] was used for modeling and running the branch-and-bound algorithm.

Algorithm 4: Step 3 of Algorithm 2: improve_solution.

Input: instance $\mathcal{S}$, current solution s, distance d, and parameter N
Output: resulting solution s and distance d

for $k \leftarrow 1$ **to** n **do** $d'_k \leftarrow d_k \leftarrow d_H(s^k, s)$
for $i \leftarrow 1$ **to** N **do**
  $b \leftarrow r$ such that $d'_r = d$ /* break ties randomly */
  for $j \leftarrow 1$ **to** m such that $s^b_j \neq s_j$ **do**
    $max \leftarrow -1$
    for $k \leftarrow 1$ **to** n **do**
      if $(k \neq b)$ **then**
        if $(s_j = s^k_j)$ **and** $(s^b_j \neq s^k_j)$ **then**
          $d_k \leftarrow d_k + 1$
        else if $(s_j \neq s^k_j)$ **and** $(s^b_j = s^k_j)$ **then**
          $d_k \leftarrow d_k - 1$
        end
      else
        $d_k \leftarrow d_k - 1$
      end
      if $(max < d_k)$ **then**
        $max \leftarrow d_k$
      end
    end
    if $(d \geq max)$ /* this is not worse */ **then**
      $d \leftarrow max$; $s_j \leftarrow s^b_j$
      for $k \leftarrow 1$ **to** n **do** $d'_k \leftarrow d_k$
    else
      for $k \leftarrow 1$ **to** n **do** $d_k \leftarrow d'_k$
    end
  end
end

4.2 Experimental Results

The B&B algorithm was executed for a set of 58 instances, from which 30 instances used the alphabet $\{A, T, G, C\}$ and 28 the alphabet $\{0, 1\}$. The maximum time allowed for each instance was one hour (after the computation of the initial linear relaxation). The results are shown in Tables 2 and 3, respectively. The columns in these tables have the following meaning. The first three columns give information about the instances. The column "sol" gives the value of the best integer solution found. When the B&B is run into completion, this means the optimal solution. Otherwise, this is the value of the best solution found by the B&B algorithm. The LB column shows the lower bound obtained after solving the root node in the B&B tree. LB is the ceil of the linear relaxation value. Column "# nodes" has the number of nodes explored in the enumeration tree and "time" has the total CPU time (in seconds) spent by the B&B. The final three columns give solution values, times (in seconds) for execution of the

instance	opt RP1	opt RP3	opt
1	90	174	174
2	121	235	235
3	151	290	291
4	182	348	348
5	212	404	404
6	238	458	459
7	89	185	186
8	120	247	247
9	151	301	303
10	179	367	369
11	211	427	433
12	240	489	492

Table 1: Comparison among linear relaxation values for formulations P1, P3, and the optimal integer solution.

heuristic algorithm, and the ratio between the heuristic solution value and the optimal integer solution value.

The experiments show that the heuristic algorithm was able to find solutions within 16% of the optimal value in less than 1 minute for instances with $\mathcal{A} = \{0, 1\}$. But when the alphabet is $\mathcal{A} = \{A, T, G, C\}$, our heuristic found solutions within 5% of an optimal value within 1 minute. Since the heuristic is mostly controlled by the number of iterations one allows it to run, one could try to get better solutions by giving it more time. For our purpose of assessing the quality of the heuristic algorithm, we have set the number of iterations equal to 100,000. We have noticed that for the binary alphabet no improvement in the value of objective function is made after 1,000 iterations. However, for the other alphabet some improvements were obtained for iterations close to the limit of 100,000.

The results for the B&B in Tables 2 and 3 show that the IP formulation P3 gives very good lower bounds on the value of an optimal solution for a CSP instance. We have noticed that applying the primal heuristic for the root node gives most of the time an optimal solution. Even though the P3 gives good lower bounds, we were not able to improve it by adding Gomory cuts, for example. This is the main reason that the number of nodes in the B&B tree is large for some instances. We have also noticed that for the tested instances there are many alternative optimal solutions and a large number of nodes in the B&B tree have the same linear relaxation value. This contributes for increasing the number of nodes to be explored.

Table 1 shows a comparison between linear relaxation values for the formulations P1 and P3. The comparison is done for the instances from 1 to 12 in Table 2.

instance			B&B				heuristic		
num	n	m	sol	LB	# nodes	time	sol	time	ratio
1	10	300	174	174	5652	10.87	181	5	1.04
2	10	400	235	235	647	2.92	239	7	1.02
3	10	500	291	290	1	0.93	301	10	1.03
4	10	600	348	348	1225	8.97	354	11	1.02
5	10	700	404	404	933	9.46	411	13	1.02
6	10	800	459	458	2579	17.52	472	15	1.03
7	15	300	186	185	1701997	4024.78	191	8	1.03
8	15	400	247	247	3025	10.17	256	12	1.05
9	15	500	303	301	201845	780.41	315	15	1.04
10	15	600	369	367	4994	22.83	375	18	1.02
11	15	700	433	427	704590	3925.63	437	20	1.01
12	15	800	492	489	616061	3877.36	500	23	1.02
13	20	300	190	190	108281	358.90	196	12	1.03
14	20	400	255	253	345031	1261.30	263	17	1.03
15	20	500	316	316	166950	671.74	327	22	1.03
16	20	600	379	378	632070	3868.59	391	24	1.03
17	20	700	442	442	3755	22.88	449	28	1.02
18	20	800	506	505	512329	3802.08	518	33	1.02
19	25	300	196	195	1405903	4004.59	204	16	1.04
20	25	400	260	259	843502	3874.79	266	21	1.02
21	25	500	322	321	658143	3812.68	334	27	1.04
22	25	600	390	389	577236	3842.66	398	30	1.02
23	25	700	454	453	501478	3786.70	462	36	1.02
24	25	800	516	516	97408	698.46	532	41	1.03
25	30	300	198	197	1222037	3975.23	203	18	1.03
26	30	400	265	264	831860	3875.11	271	26	1.02
27	30	500	328	327	589934	3845.15	336	32	1.02
28	30	600	394	393	456213	3765.45	405	37	1.03
29	30	700	459	458	406539	3824.78	468	43	1.02
30	30	800	525	524	367610	3752.77	542	49	1.03

Table 2: Results for instances using the alphabet $\mathcal{A} = \{A, T, G, C\}$.

instance			B&B				heuristic		
num	n	m	sol	LB	# nodes	time	sol	time	ratio
31	10	400	156	155	2687979	4141.82	171	5	1.10
32	10	500	190	189	2184136	4189.93	190	5	0.00
33	10	600	229	228	1733818	4485.28	247	7	1.08
34	10	800	305	303	1482246	4006.00	306	9	1.003
35	15	300	122	121	2806838	4128.27	130	5	1.07
36	15	400	159	158	2166362	4156.88	161	7	1.01
37	15	500	202	200	1772538	4071.95	213	9	1.05
38	15	600	240	239	1503438	3967.02	246	11	1.03
39	15	700	278	277	1318088	4101.48	323	14	1.16
40	15	800	325	324	1092610	3911.65	333	15	1.02
41	20	300	126	125	2386388	4063.52	133	8	1.06
42	20	400	165	164	1872385	4063.76	191	11	1.16
43	20	500	207	206	1454067	3963.46	218	14	1.05
44	20	600	247	246	1203460	3918.93	265	16	1.08
45	20	700	291	290	1080569	3913.32	311	19	1.07
46	20	800	331	330	944268	3870.64	359	21	1.08
47	25	300	131	130	1943166	4002.80	145	10	1.11
49	25	400	173	172	1541728	3975.91	182	14	1.05
50	25	500	211	211	17158	56.25	228	18	1.08
51	25	600	255	254	1041537	3881.25	271	19	1.06
52	25	700	300	299	893404	3948.62	314	23	1.05
53	25	800	337	336	786798	3903.74	351	26	1.04
54	30	300	133	132	1674260	4026.07	150	13	1.13
55	30	400	174	172	1367616	4002.25	184	16	1.06
56	30	500	217	216	1073655	4029.87	242	22	1.12
57	30	600	263	262	874222	4025.71	296	26	1.13
58	30	700	301	299	773934	4015.73	316	27	1.05

Table 3: Results for instances using the alphabet $\mathcal{A} = \{0, 1\}$.

5. Concluding Remarks

In this paper we introduce three IP models for the CSP. Our goal is to solve the problem exactly by using IP algorithmic techniques. Theoretical developments based on those models are made. Our experiments on randomly generated instances have shown that the new heuristic algorithm that we have proposed is capable of finding good upper bounds and by using them in conjunction with the branch-and-bound algorithm, it is possible to speed up the performance of this algorithm.

The B&B here introduced was able to solve to optimality instances of size up to $n = 30$ and $m = 800$ with alphabet $\mathcal{A} = \{0, 1\}$ and $\mathcal{A} = \{A, T, G, C\}$. As a future work we intend to imporve the perfomance of the B&B algorithm.

References

[1] P. Berman, D. Gumucio, R. Hardison, W. Miller, and N. Stojanovic. A linear-time algorithm for the 1-mismatch problem. In *WADS'97*, 1997.

[2] Dash Optimization Inc. *Xpress-Optimizer Reference Manual*, 2003.

[3] M. Frances and A. Litman. On covering problems of codes. *Theor. Comput. Syst.*, 30:113–119, 1997.

[4] L. Gasieniec, J. Jansson, and A. Lingas. Efficient approximation algorithms for the hamming center problem. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pages S905–S906, 1999.

[5] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for closest string and related problems. In *In Proceedings of the 12th Annual International Symposium on Algorithms and Computation (ISAAC 2001)*, volume 2223 of *Lecture Notes in Computer Science*, pages 441–452. Springer Verlag, 2001.

[6] G. Hertz and G. Stormo. Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In Lim and Cantor, editors, *Proc. 3rd Int'l Conf. Bioinformatics and Genome Research*, pages 201–216. World Scientific, 1995.

[7] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pages 633–642, 1999. to appear in Information and Computation.

[8] E.K. Lee and J.E. Mitchell. Integer programming: Branch and bound methods. In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, volume 2, pages 509–519. Kluwer Academic Publishers, 2001.

[9] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. *Proceedings of the Thirty First Annual ACM Symposium on Theory of Computing, Atlanta*, pages 473–482, 1999.

[10] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.

[11] S. Park and K. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31:1192–1201, 1988.

[12] S. Rajasekaran, Y. Hu, J. Luo, H. Nick, P.M. Pardalos, S. Sahni, and G. Shaw. Efficient algorithms for similarity search. *Journal of Combinatorial Optimization*, 5:125–132, 2001.

[13] S. Rajasekaran, H. Nick, P.M. Pardalos, S. Sahni, and G. Shaw. Efficient algorithms for local alignment search. *Journal of Combinatorial Optimization*, 5:117–124, 2001.

[14] S. Roman. *Coding and Information Theory*, volume 134 of *Graduate Texts in Mathematics*. Springer-Verlag, 1992.

[15] G. Stormo and G.W. Hartzell III. Identifying protein-binding sites from unaligned DNA fragments. *Proc. Natl. Acad. Sci. USA*, 88:5699–5703, 1991.