INFORMS Journal on Computing
Vol. 16, No. 4, Fall 2004, pp. 419–429
ISSN 0899-1499 | EISSN 1526-5528 | 04 | 1604 | 0419

DOI 10.1287/ijoc.1040.0090
© 2004 INFORMS

Optimal Solutions for the Closest-String Problem via Integer Programming

Cláudio N. Meneses
Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, Florida 32611, USA, claudio@ufl.edu

Zhaosong Lu
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0205, USA, zhaosong@isye.gatech.edu

Carlos A. S. Oliveira, Panos M. Pardalos
Department of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, Florida 32611, USA {oliveira@ufl.edu, pardalos@ufl.edu}

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 center 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 an 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 (IP) 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.

Key words: closest-string problem; computational biology; mathematical programming; branch-and-bound algorithms
History: Accepted by Harvey J. Greenberg, Guest Editor; received February 2004; revised February 2004, March 2004; accepted April 2004.

1. Introduction

In many problems occurring in computational biology, there is a need to compare and find 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 that minimizes the number of differences among elements in a given set $\mathcal{S}$ of strings. The distance here is defined as the Hamming distance between t and each $s^i \in \mathcal{S}$. 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 (Gasieniec et al. 1999; Hertz and Stormo 1995; Lanctot et al. 2003; Rajasekaran et al. 2001a, b; Stormo and Hartzell 1991) 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, one needs to design genetic drugs with structures similar to a set of existing sequences of RNA (Lanctot et al. 2003). As another example, in coding theory (Roman 1992) the objective is to find sequences of characters that are closest to some given set of strings. This is useful in determining the best way to encode a set of messages.

The CSP is known to be NP-hard (Frances and Litman 1997). Even though there are good approximation algorithms for this problem, including a polynomial-time approximation scheme (PTAS) (Li et al. 2002), and exact algorithms for instances with constant difference (Berman et al. 1997), 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.

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 $|s|$ is the number of elements in the character sequence that composes the string. Thus, if $s = (s_1, \ldots, s_m)$, then $|s| = m$.

We define the *Hamming distance* $d_H(a, b)$ between two strings a and b such that $|a| = |b|$ as the number of positions in which they differ. Let us define the predicate function $\mathcal{E}: \mathcal{A} \times \mathcal{A} \to \{0, 1\}$ as $\mathcal{E}(x, y) = 1$ if and only if $x \neq y$. Thus, we have $d_H(a, b) = \sum_{i=1}^{|a|} \mathcal{E}(a_i, b_i)$. For instance, d_H("CATCC", "CTTGC") $= 2$.

Let $\mathcal{A}^k$ be the set of all strings s over the alphabet $\mathcal{A}$ with length $|s| = k$. 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 center 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 $d = 4$.

The recent development of computational-biology applications has required the study of optimization problems over sequences of characters. Some early development was done using statistical methods (Hertz and Stormo 1995). Li et al. (1999), the combinatorial nature of the CSP was initially explored, and it was proved to be NP-hard. Li et al. (1999) also described some applications in molecular biology where this problem has been very useful. In Gramm et al. (2001) it was shown that for a fixed value of d it is possible to solve the problem in polynomial time, using techniques of fixed parameter complexity (see, e.g., Downey and Fellows 1999). Approximation algorithms have also been used to give near optimal solutions for the CSP. In Lanctot et al. (2003) for example, an algorithm with performance guarantee of $(4/3)(1+\epsilon)$, for any small constant $\epsilon > 0$, is presented and analyzed, with a similar result appearing also in Gasieniec et al. (1999). A PTAS for the CSP was presented in Li et al. (2002).

In this paper we are interested in optimal solutions for the general case of the CSP where the maximum distance between the center string and the set of strings is variable. With this objective we propose three integer-programming formulations and explore properties of these formulations (the third formulation has been independently proposed in Ben-Dor et al. 1997 and Li et al. 2002). We also show computational results of a branch-and-bound algorithm that uses the linear relaxation of one of the proposed formulations. 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 to optimality CSP instances of moderate size.

This paper is organized as follows. In §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 §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. Also in this section, we present a new heuristic algorithm for the CSP. In §4 we show computational results obtained by the proposed methods. Finally, concluding remarks are given in §5.

2. Integer-Programming Formulations

In this section we present three IP formulations and show that they correctly describe the CSP. Our main result in this section states that the third formulation is the strongest of them. In §2.1 we introduce definitions and notations. The IP formulations are given in §§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 $|s^i| = m$, 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 $x^i \in \mathbf{Z}^m$. 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 alphabet $\{a, \ldots, z\}$ and the assignment $\pi(a) = 1$, $\pi(b) = 2, \ldots, \pi(z) = 26$ are used, unless otherwise stated. Now, let $\phi_\pi: \mathcal{A}^m \to \mathbf{Z}^m$ be a function mapping strings 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 throughout the paper.

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 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.

$$\text{P1:} \quad \min \ d \tag{1}$$

$$\text{s.t.} \ \sum_{k=1}^{m} z_k^i \leq d \quad i = 1, \ldots, n \tag{2}$$

$$t_k - x_k^i \leq K z_k^i \quad i = 1, \ldots, n; \ k = 1, \ldots, m \tag{3}$$

$$x_k^i - t_k \leq K z_k^i \quad i = 1, \ldots, n; \ k = 1, \ldots, m \tag{4}$$

$$z_k^i \in \{0, 1\} \quad i = 1, \ldots, n; \ k = 1, \ldots, m \tag{5}$$

$$d \in \mathbf{Z}_+ \tag{6}$$

$$t_k \in \mathbf{Z} \quad k = 1, \ldots, m, \tag{7}$$

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 because 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) through (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 constraints in* P1.

Proof. Let $\mathcal{S}_\mathcal{T}$ be an instance of the CSP. It is clear that any vector $x \in \mathbf{Z}^m$ such that $x_i \in \{1, \ldots, |\mathcal{A}|\}$ for $i \in \{1, \ldots, m\}$, 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) through (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

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

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 $s^i \in \mathcal{S}$, $|x_k^i - x_k| \leq \text{diam}(\Omega_k)$.*

Proof. Given an optimum solution t, apply Proposition 2. Then, we find another optimum solution t' such that each character t'_k appears in position k of some string $s^i \in \mathcal{S}$. Applying to 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:

$$t_k - x_k^i \le \operatorname{diam}(\Omega_k) z_k^i \quad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{8}$$

$$x_k^i - t_k \le \operatorname{diam}(\Omega_k) z_k^i \quad i = 1, \ldots, n;\ k = 1, \ldots, m. \tag{9}$$

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 $\mathcal{S} = \{\text{“}ABC\text{”}, \text{“}DEF\text{”}\}$, the optimum solution has $d = 2$. An optimal string is $t = \text{“}EBF\text{”}$, which has distance $d = 2$, although it is not feasible for P1A. However, according to Proposition 2, any optimum 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 a strengthening of P1, and has $n + 3nm + 1$ constraints and $m + nm + 1$ variables.

2.3. Second IP Formulation

In this section 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 $i \le m$, and return the resulting string t.

Alternatively, this process can be thought of 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 1.

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 vertex F to vertex D. For instance, taking the directed path $(F, 4, 9, 6, 6, 1, 13, D)$ in the graph shown in Figure 1, and discarding nodes F and D, we obtain the feasible solution $(4, 9, 6, 6, 1, 13)$, which corresponds to string *“diffam”*.

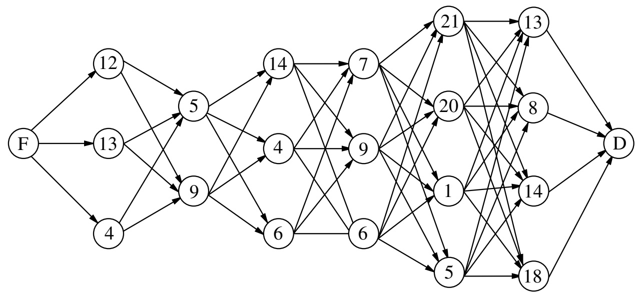

Figure 1 **Directed Graph $G = (V, A)$ for Strings in Example 1**

One can think of enumerating all directed paths from F to D and choose one with minimum Hamming distance to all strings in $\mathcal{S}_\mathcal{T}$. However, the number of such paths is given by $\prod_{k=1}^m |V_k|$. Because $|V_k|$ 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

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

Then, another IP formulation is defined in the following way:

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

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

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

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

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

$$x_k^i - t_k \le \operatorname{diam}(\Omega_k) z_k^i \quad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{15}$$

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

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

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

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

The additional equalities (11) ensure that only one vertex 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 optimum 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 optimum solution of P2 has the same objective value as the optimum 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 optimum value of* RP1A *and* z_2^* *is the optimum value of* RP2, *then* $z_1^* = z_2^*$.

Proof. We know that P2 $\subset$ P1A, which implies RP2 $\subseteq$ RP1A. Thus, $z_1^* \le z_2^*$. Then, it suffices to prove that $z_2^* \le z_1^*$. 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) through (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 $k \in \{1, \dots, m\}$. 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 $j \in V_k \backslash \{a, b\}$. □

We note that the number of variables and constraints in P2 are given by $m + nm + \sum_{k=1}^m |V_k| + 1$ and $3nm + 2m + n + \sum_{k=1}^m |V_k| + 1$.

2.4. Third IP Formulation

In this subsection we propose another IP formulation for the CSP using the idea in Proposition 2. This formulation has been proposed, independently, in Ben-Dor et al. (1997) and Li et al. (2002). We use the same definition of variables $v_{j,k}$ given in §2.3.

The third IP formulation is given by

$$\text{P3:} \quad \min \ d \qquad (20)$$

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

$$m - \sum_{j=1}^m v_{s_j^i, j} \le d \quad i = 1, \dots, n \qquad (22)$$

$$v_{j,k} \in \{0,1\} \quad j \in V_k,\ k = 1, \dots, m \qquad (23)$$

$$d \in \mathbf{Z}_+. \qquad (24)$$

Inequalities (21) guarantee that only one vertex in each V_k is selected. Inequalities (22) say that if a vertex in a string s^i is not in a solution t, then that vertex will contribute to increasing 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 optimum 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 $\mathcal{S}$ because any $x \in \mathbf{Z}^m$ such that $x_i \in \{1, \dots, |\mathcal{A}|\}$, for $i \in \{1, \dots, m\}$, is feasible for $\mathcal{S}$. Now, if t defines a feasible solution to $\mathcal{S}$ satisfying Proposition 2, then $P = (F, t_1, \dots, t_m, D)$ defines a directed path from F to D in the graph G, constructed from $\mathcal{S}$ as described in the previous subsection. 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 $\sum_{j=1}^m v_{s_j^i, j}$ is at most m. Hence, $d \ge 0$, 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 optimum 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 $\sum_{k=1}^m |V_k| + 1$ and $m + n + \sum_{k=1}^m |V_k| + 1$, respectively. In the next theorem, we determine the relationship between P3 and the previous formulations.

Theorem 7. *The IP formulations* P1 *and* P3 *satisfy* P3 $\subseteq$ P1, *where* $A \subseteq B$ *means that set* A *is contained in set* B.

Proof. We show that any feasible solution for P3 is also feasible for P1. If d, $v_{j,k}$, for $j \in V_k$, $k = 1, \dots, m$, is a feasible solution for P3, we can define a feasible solution d, t_k, z_k^i, where $i \in \{1, \dots, n\}$, $k \in \{1, \dots, m\}$, for P1 in the following way. Set

$$t_k = \sum_{j \in V_k} j v_{j,k} \quad \text{for } k \in \{1, \dots, m\}, \quad \text{and}$$

$$z_k^i = 1 - v_{s_k^i, k} \quad \text{for } k \in \{1, \dots, m\},\ i = \{1, \dots, n\}.$$

Constraints (5) are automatically satisfied by this definition. From the second constraint of P3, we have

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

Therefore, 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 $t_k - x_k^i < 0$, 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 P1 are satisfied by the solution defined above. □

Note that from the point of view of the feasible set, P3 is similar to P2 because 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 §4.2.

3. Implementation Issues

Using the formulations described in the previous section, we developed a branch-and-bound (B&B) algorithm 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

In this paper a B&B is proposed for solving the CSP. An introduction to the technique is given by Lee and Mitchell (2001). The first important decision when applying B&B 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. The computational effort to solve the CSP with P3 is smaller because there are fewer constraints and variables, and the formulation cuts off many solutions that are not needed to find the optimum. Taking this fact into consideration, the relaxation of P3 was used on the B&B procedure.

```
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
$d \leftarrow \max_{i \in \{1,\ldots,n\}}(m - \sum_{j=1}^{m} v_{s_j^i,j})$
```

Algorithm 1 Primal Heuristic for Formulation P3

An efficient primal heuristic is important for the success of a B&B implementation because 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 the 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 $O(nm)$.

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, in the algorithm used, the next node to be explored in the enumeration tree is the one with the smallest linear relaxation value (also known as the *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 because 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

```
  Input: instance $\mathcal{S}$
  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$(s, d, N)$
```

Algorithm 2 Generate Upper Bound for the CSP

the strings in $\mathcal{S}$ and modifying it until a new locally optimal solution is found.

In the first step, the algorithm searches for a solution $s \in \mathcal{S}$ that is closest to all other strings in $\mathcal{S}$. 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 $i \in \{1, \ldots, n\}$, 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 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

```
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
    $b \leftarrow i$ such that $d_H(s^i, s) = d$ /* 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 $n$ such that $k \neq b$ do
            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 $d_k \leftarrow d_k - 1$
            if $(max < d_k)$ then $max \leftarrow d_k$
        end
        if $d \geq max$ /* this is not worse */ then
            $d \leftarrow max$; $t_j \leftarrow s^b_j$
            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
```

Algorithm 3 Step 3 of Algorithm 2: improve_solution

details of the local search step are presented in Algorithm 3.

We now analyze the computational complexity of Algorithm 2.

Table 1 Summary of Results for the Alphabet with Two Characters

Instance		LP				IP			
n	m	Min	Avg	Max	Time	Min	Avg	Max	Time
10	300	111.14	112.33	113.90	0.10	112	113.25	115	1,800.10
10	400	148.20	150.60	152.80	0.11	149	151.50	154	1,800.33
10	500	186.60	189.55	192.80	0.14	187	190.50	194	2,700.07
10	600	221.70	224.15	228.50	0.16	223	225.00	229	900.84
10	700	259.80	262.95	266.80	0.18	260	263.75	267	1,800.24
10	800	300.00	301.55	303.20	0.21	301	302.25	304	901.03
15	300	118.30	119.37	120.50	0.11	119	120.25	122	915.31
15	400	157.55	158.53	159.01	0.12	158	159.50	160	941.72
15	500	195.55	196.99	197.49	0.13	197	198.00	199	1,801.22
15	600	236.35	237.59	238.93	0.14	237	238.50	240	900.23
15	700	275.94	277.75	279.64	0.18	277	278.50	280	14.06
15	800	315.96	318.37	320.36	0.19	317	319.25	321	62.00
20	300	123.19	124.02	125.02	0.12	124	125.00	126	903.03
20	400	164.68	166.08	168.07	0.12	166	167.25	169	2,037.49
20	500	204.05	206.35	208.41	0.14	205	207.25	209	907.32
20	600	246.08	247.56	248.68	0.17	247	248.75	250	2,702.18
20	700	284.70	286.50	288.98	0.18	286	287.50	290	1,042.29
20	800	326.12	328.88	331.47	0.21	327	330.00	333	1,852.38
25	300	126.32	127.25	128.93	0.12	128	128.75	130	2,701.88
25	400	169.09	169.90	171.31	0.15	170	171.00	172	1,821.56
25	500	210.58	210.78	211.24	0.17	212	212.00	212	2,790.50
25	600	251.40	252.11	253.54	0.18	253	253.60	255	3,600.26
25	700	294.93	295.51	295.84	0.21	296	296.75	297	2,721.62
25	800	334.91	336.62	338.34	0.23	336	338.00	340	3,600.07
30	300	128.97	130.34	131.60	0.13	130	131.50	133	955.92
30	400	171.36	172.39	173.73	0.16	173	173.75	175	3,600.18
30	500	213.20	215.89	217.84	0.18	215	217.50	219	3,600.12
30	600	257.26	258.06	259.09	0.21	259	259.75	261	3,600.07
30	700	299.65	300.51	301.20	0.26	301	302.00	303	2,988.07
30	800	341.48	342.52	342.90	0.28	343	343.75	344	3,600.20

Theorem 8. *Algorithm* 2 *has complexity* $O(nmN)$, *for* $N \geq n$.

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

4. Computational Experiments

We now present the computational experiments carried out with the algorithms described in §3. Initially, we describe the set of instances used in the tests. Then, in §4.2 the results obtained by the B&B and by the proposed heuristic are discussed.

4.1. Instances and Test Environment

Three classes of instances were tested. In the first class, the alphabet $\mathcal{A}$ is the set $\{0, 1\}$, representing binary strings. The second class of instances uses an alphabet with four characters, while the third class uses an alphabet with 20 characters. This choice of the last two alphabets was motivated by real applications on analysis of DNA and amino-acid sequences, respectively.

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 (Park and Miller 1988), with parameters 16,807 (multiplier) and $2^{31} - 1$ (prime number).

All tests were executed on a Pentium 4 CPU with speed of 2.80 GHz and 512 MB of RAM, under WindowsXP. The heuristic algorithm was implemented in the C++ language, and CPLEX 8.1 (ILOG 2003) was used to run the B&B.

4.2. Experimental Results

The B&B algorithm was executed over a set of 360 instances, with 120 instances for each of the alphabets. Four instances were generated for each entry

Table 2 Summary of Results for the Alphabet with Four Characters

Instance		LP				IP			
n	m	Min	Avg	Max	Time	Min	Avg	Max	Time
10	300	173.20	174.85	176.30	0.21	174	175.50	177	0.12
10	400	232.10	233.50	235.60	0.29	233	234.00	236	0.12
10	500	287.90	289.23	291.90	0.35	288	289.50	292	0.34
10	600	346.40	348.62	350.50	0.45	347	349.25	351	0.18
10	700	404.80	407.07	409.30	0.53	405	407.50	410	1.84
10	800	462.90	466.15	468.60	0.67	463	466.50	469	0.60
15	300	182.36	183.57	186.09	0.25	183	184.25	187	6.02
15	400	242.39	244.89	246.16	0.32	243	245.50	247	3.46
15	500	305.50	306.11	306.87	0.43	306	306.50	307	44.98
15	600	365.00	367.72	370.87	0.54	365	368.00	371	18.39
15	700	425.69	428.07	431.12	0.75	426	428.50	432	21.78
15	800	488.93	490.00	492.73	0.89	489	490.50	493	34.39
20	300	187.11	188.93	190.76	0.27	190	190.25	191	1,170.98
20	400	251.55	252.22	253.25	0.39	252	253.00	254	908.56
20	500	313.42	315.51	317.96	0.52	314	316.25	319	901.44
20	600	377.54	379.64	380.84	0.68	378	380.00	381	1,271.88
20	700	441.10	441.88	443.01	0.93	442	442.75	444	940.21
20	800	503.72	505.05	506.25	1.16	505	505.75	507	1,714.42
25	300	192.18	193.99	195.47	0.31	195	195.75	196	2,741.15
25	400	257.67	258.55	259.73	0.47	259	259.75	261	2,700.30
25	500	320.13	322.52	324.11	0.60	321	323.25	325	741.62
25	600	385.91	387.00	388.14	0.83	387	388.00	389	1,805.16
25	700	449.12	451.30	452.30	1.09	450	452.25	453	907.61
25	800	514.06	515.76	518.65	1.40	515	516.75	520	1,254.19
30	300	196.51	197.10	197.73	0.36	198	198.25	199	3,223.68
30	400	261.09	262.27	263.33	0.47	262	263.25	264	1,852.67
30	500	325.42	328.23	329.52	0.70	326	329.25	331	2,215.67
30	600	392.01	392.89	393.55	0.99	393	394.25	395	2,700.30
30	700	458.05	458.56	459.10	1.26	459	459.75	460	1,803.28
30	800	521.24	522.55	524.68	1.62	522	523.50	526	2,337.20

Table 3 Summary of Results for the Alphabet with 20 Characters

Instance		LP				IP			
n	*m*	Min	Avg	Max	Time	Min	Avg	Max	Time
10	300	235.20	236.10	237.00	0.20	236	236.50	237	0.15
10	400	311.40	312.60	314.30	0.28	312	313.25	315	0.17
10	500	390.40	391.60	393.70	0.36	391	392.00	394	0.23
10	600	468.10	470.14	471.80	0.45	469	471.40	472	0.29
10	700	546.90	547.45	548.60	0.62	547	547.75	549	0.51
10	800	625.20	626.12	627.50	0.79	626	626.75	628	0.43
15	300	244.47	245.69	246.47	0.31	245	246.25	247	0.23
15	400	326.53	327.15	327.47	0.43	327	327.75	328	0.33
15	500	409.33	409.95	411.33	0.57	410	410.50	412	0.38
15	600	491.67	492.50	493.53	0.80	492	493.00	494	0.56
15	700	572.07	574.24	577.47	1.09	573	575.00	578	0.67
15	800	655.13	655.97	657.27	1.43	656	656.75	658	0.90
20	300	249.90	251.00	252.05	0.43	250	251.50	253	2.77
20	400	335.40	335.66	335.75	0.62	336	336.00	336	0.93
20	500	418.40	419.12	419.90	0.84	419	419.50	420	0.85
20	600	502.60	503.25	504.00	1.25	503	503.50	504	3.95
20	700	585.00	585.64	586.60	1.71	585	586.00	587	4.37
20	800	671.30	671.64	672.05	1.66	672	672.25	673	1.94
25	300	254.60	255.12	256.11	0.49	255	255.75	257	0.58
25	400	340.64	340.86	341.04	0.77	341	341.25	342	189.59
25	500	425.32	426.10	426.72	1.15	426	426.50	427	40.12
25	600	509.80	510.73	511.48	1.74	510	511.00	512	19.79
25	700	595.24	596.28	597.04	1.48	596	596.75	598	50.71
25	800	680.88	681.88	682.96	1.79	681	682.00	683	863.77
30	300	258.29	258.90	259.57	0.66	259	259.50	260	0.58
30	400	344.00	344.54	345.27	1.02	344	345.00	346	49.04
30	500	430.43	430.79	431.07	1.71	431	431.25	432	629.32
30	600	516.73	517.00	517.50	2.10	517	517.25	518	155.07
30	700	601.97	603.00	603.50	1.61	603	603.75	604	902.10
30	800	688.30	689.26	690.60	2.17	689	689.75	691	198.94

in Tables 1 through 3, and the results represent the average of the obtained values. The maximum time allowed for each instance was one hour (after the computation of the initial linear relaxation). The columns in these tables have the following meaning. The first two columns give information about the instances: the number of strings (n) and their length (m). The columns labeled "LP" give the minimum, average, and maximum LP value, as well as the average running time (in seconds) for the linear relaxation of formulation P3. The columns labeled "IP" give the minimum, average, and maximum IP value, as well as the average running time (in seconds) for the B&B.

Notice that some large instances presented in Table 1 could be solved in a small amount of CPU time. In particular, this happened for instances with $n = 15$ and $m = 700, 800$. According to our investigation, these instances (which were also generated randomly) represented configurations of the CSP that were easier to solve. However, we were not able to identify the properties that make these instances easier to solve. One can conclude that even for random instances, the computation time is not only directly proportional to instance size, but also to its structure.

Table 4 shows the results of experiments with the heuristic. For each instance, the following information is presented for alphabets with 2, 4, and 20 characters. The column labeled "val" gives the average solution value and the next column gives the running time (in seconds) for the heuristic algorithm. The third column gives the relative gap between the heuristic solution and the IP solution, calculated as h/i, where h is the average heuristic value and i is the average IP solution value. The experiments show that the heuristic algorithm was able to find solutions within 15% of the optimal value in less than one minute for instances with the binary alphabet. But when the alphabet has four characters, our heuristic found solutions within 4% of an optimal value within one minute, and within the maximum of 7% for the alphabet with 20 characters. Because 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 10,000.

Table 4 **Summary of Results for the Heuristic**

Instance		2 Characters			4 Characters			20 Characters		
n	*m*	Val	Time	Gap	Val	Time	Gap	Val	Time	Gap
10	300	124.50	0.00	1.10	178.25	0.00	1.02	249.50	0.00	1.05
10	400	156.25	0.00	1.03	237.50	0.00	1.01	329.75	0.25	1.05
10	500	194.50	0.00	1.02	295.00	1.00	1.02	413.25	1.00	1.05
10	600	233.75	0.00	1.04	353.75	1.00	1.01	495.75	1.00	1.05
10	700	300.25	0.50	1.14	413.75	1.00	1.02	577.00	1.00	1.05
10	800	309.75	0.50	1.02	473.25	1.00	1.01	656.00	1.00	1.05
15	300	138.50	0.00	1.15	188.50	1.00	1.02	261.75	1.00	1.06
15	400	171.25	0.00	1.07	251.50	1.00	1.02	348.50	1.00	1.06
15	500	207.50	0.50	1.05	316.25	1.00	1.03	436.50	1.00	1.06
15	600	256.25	1.00	1.07	378.50	1.00	1.03	523.00	2.00	1.06
15	700	309.75	1.00	1.11	440.25	2.00	1.03	609.25	2.00	1.06
15	800	346.25	1.00	1.08	503.75	2.00	1.03	695.25	2.00	1.06
20	300	134.25	0.00	1.07	195.50	1.00	1.03	268.50	1.00	1.07
20	400	178.25	1.00	1.07	261.50	1.00	1.03	359.00	1.00	1.07
20	500	230.00	1.00	1.11	325.50	2.00	1.03	447.00	2.00	1.07
20	600	262.25	1.00	1.05	391.00	2.00	1.03	536.50	2.00	1.07
20	700	316.00	1.50	1.10	456.75	3.00	1.03	624.75	3.00	1.07
20	800	354.75	2.00	1.07	521.75	3.00	1.03	713.50	3.00	1.06
25	300	135.25	1.00	1.05	202.50	1.00	1.03	273.25	1.00	1.07
25	400	192.50	1.00	1.13	267.50	2.00	1.03	363.50	2.00	1.07
25	500	229.75	1.00	1.08	334.00	2.50	1.03	453.75	2.00	1.06
25	600	270.80	2.00	1.07	399.25	3.00	1.03	544.25	3.00	1.07
25	700	320.50	2.00	1.08	467.00	3.00	1.03	635.75	3.25	1.07
25	800	373.00	2.00	1.10	533.25	4.00	1.03	727.75	4.00	1.07
30	300	141.50	1.00	1.08	204.25	1.00	1.03	276.50	1.50	1.07
30	400	187.00	1.00	1.08	272.75	2.25	1.04	367.75	2.00	1.07
30	500	237.25	2.00	1.09	339.25	3.00	1.03	459.00	3.00	1.06
30	600	291.00	2.00	1.12	405.50	3.00	1.03	551.25	3.00	1.07
30	700	329.25	2.50	1.09	475.00	4.00	1.03	641.75	4.00	1.06
30	800	381.00	3.00	1.11	540.25	5.00	1.03	733.25	5.00	1.06

The results for the B&B in Tables 1 through 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 an optimal solution most of the time. 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 is the main reason why the number of nodes in the B&B tree is large for some instances.

Table 5 **Summary of Results for the McClure Instances, over the Alphabet with 20 Characters**

Instance			LP		IP		Heuristic		
Name	*n*	*m*	Val	Time	Val	Time	Val	Time	Gap
mc582.10.seq	10	141	71.83	0.08	72	0.52	78	1	1.08
mc582.12.seq	12	141	94.83	0.38	95	0.16	100	1	1.05
mc582.6.seq	6	141	74.40	0.28	75	0.08	78	1	1.04
mc586.6.seq	6	100	96.52	0.16	97	0.28	100	1	1.03
mc586.10.seq	10	98	75.82	0.14	76	0.27	80	1	1.05
mc586.12.seq	12	98	96.56	0.17	97	0.13	101	1	1.04

Table 5 shows results for some instances over the alphabet with 20 characters. These instances are taken from the McClure data set (McClure et al. 1994), a set of protein sequences frequently used to test string-comparison algorithms. For each of the instances considered, the size of the strings (m) is equal to the length of the smallest string in the set (this was necessary because the McClure instances have strings of different lengths). We removed the last characters for strings with length greater than the minimum.

5. Concluding Remarks

In this paper we have introduced three IP models for the CSP. Our goal was to solve the problem exactly by using IP algorithmic techniques. Theoretical developments based on those models were made. Our experiments on randomly generated instances have shown that the new heuristic algorithm we have proposed is capable of finding good upper bounds, and by using them in conjunction with the B&B, it is possible to speed up the performance of this algorithm. The B&B introduced here was able to solve to optimality instances of size up to $n = 30$ and $m = 800$ with alphabets with 2, 4, and 20 characters.

The computational results suggest that the difficulty of the problem does not depend only on its size but also on its structure and other unknown properties. It is an interesting research problem to identify all parameters that characterize the computational complexity of the CSP.

Acknowledgments

This work was partially supported by NIH and NSF grants. The first author was supported by the Brazilian Federal Agency for Post-Graduate Education (CAPES), Grant No. 1797-99-9. The authors are grateful for the suggestions of referees, which helped improve the presentation.

References

Ben-Dor, A., G. Lancia, J. Perone, R. Ravi. 1997. Banishing bias from consensus sequences. A. Apostolico, J. Hein, eds. *Proc. Eighth Annual Sympos. Combin. Pattern Matching, Aarhus, Denmark, Lecture Notes in Computer Science*, No. 1264. Springer-Verlag, Heidelberg, Germany, 247–261.

Downey, R. G., M. R. Fellows. 1999. *Parameterized Complexity*. Springer-Verlag, Heidelberg, Germany.

Frances, M., A. Litman. 1997. On covering problems of codes. *Theoret. Comput. System* **30** 113–119.

Gasieniec, L., J. Jansson, A. Lingas. 1999. Efficient approximation algorithms for the Hamming center problem. *Proc. Tenth ACM-SIAM Sympos. Discrete Algorithms, Baltimore, Maryland*, Society for Industrial and Applied Mathematics, Philadelphia, PA, S905–S906.

Gramm, J., R. Niedermeier, P. Rossmanith. 2001. Exact solutions for closest string and related problems. *Proc. Twelfth Annual Internat. Sympos. Algorithms Comput. (ISAAC 2001), Lecture Notes in Computer Science*, No. 2223. Springer-Verlag, Heidelberg, Germany, 441–452.

Hertz, G., G. Stormo. 1995. Identification of consensus patterns in unaligned DNA and protein sequences: A large-deviation statistical basis for penalizing gaps. Lim, Cantor, eds. *Proc. Third Internat. Conf. Bioinformatics Genome Res.* World Scientific, Singapore, 201–216.

ILOG Inc. 2003. *CPLEX 8.1 User's Manual*. ILOG, Incline Village, NV, USA.

Lanctot, K., M. Li, B. Ma, S. Wang, L. Zhang. 2003. Distinguishing string selection problems. *Inform. Comput.* **185** 41–55.

Lee, E., J. Mitchell. 2001. Integer programming: Branch and bound methods. C. Floudas, P. Pardalos, eds. *Encyclopedia of Optimization*, Vol. 2. Kluwer Academic Publishers, Dordrecht, Netherlands, 509–519.

Li, M., B. Ma, L. Wang. 1999. Finding similar regions in many strings. *Proc. Thirty First Annual ACM Sympos. Theory Comput.* ACM Press, Atlanta, GA, 473–482.

Li, M., B. Ma, L. Wang. 2002. On the closest string and substring problems. *J. ACM* **49** 157–171.

McClure, M., T. Vasi, W. Fitch. 1994. Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evol.* **11** 571–592.

Park, S., K. Miller. 1988. Random number generators: Good ones are hard to find. *Comm. ACM* **31** 1192–1201.

Rajasekaran, S., Y. Hu, J. Luo, H. Nick, P. Pardalos, S. Sahni, G. Shaw. 2001a. Efficient algorithms for similarity search. *J. Combin. Optim.* **5** 125–132.

Rajasekaran, S., H. Nick, P. Pardalos, S. Sahni, G. Shaw. 2001b. Efficient algorithms for local alignment search. *J. Combin. Optim.* **5** 117–124.

Roman, S. 1992. *Coding and Information Theory*. *Graduate Texts in Mathematics*, No. 134. Springer-Verlag, Heidelberg, Germany.

Stojanovic, N., P. Berman, D. Gumucio, R. Hardison, W. Miller. 1997. A linear-time algorithm for the 1-mismatch problem. *Workshop on Algorithms and Data Structures, Halifax, Nova Scotia, Canada, Lecture Notes in Computer Science*, No. 1272. Springer-Verlag, Heidelberg, Germany, 126–135.

Stormo, G., G. Hartzell III. 1991. Identifying protein-binding sites from unaligned DNA fragments. *Proc. National Acad. Sci. USA* **88** 5699–5703.