Selected String Problems

42

Christian Blum and Paola Festa

Contents

Abstract

This chapter overviews some string selection and comparison problems, with special emphasis on the optimization and operational research perspective. It also proposes a simple and efficient ILP-based heuristic that can be used for any of the considered problems.

C. Blum (✉)
Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC), Bellaterra, Spain
e-mail: christian.blum@iiia.csic.es

P. Festa
Department of Mathematics and Applications “Renato Caccioppoli”, University of Napoli FEDERICO II, Napoli, Italy
e-mail: paola.festa@unina.it

R. Martí et al. (eds.), *Handbook of Heuristics*,
https://doi.org/10.1007/978-3-319-07124-4_58

Keywords
Bio-informatics · Optimization · String problems · Strings consensus

Introduction to String Problems

Among string selection and comparison problems, there is a class of problems known as *strings consensus*, where a finite set of strings is given and one is interested in finding their consensus, i.e., a new string that agrees as much as possible with all the given strings. In other words, the objective is to determine a string called consensus, because it represents—in some way—all the given strings. The idea of representation and of being in consensus can be related to several different objectives listed in the following:

(i) the consensus is a new string whose total distance from all given strings is minimum (*closest string problem*);
(ii) the consensus is a new string close to most of the given strings (*close to most string problem*);
(iii) the consensus is a new string whose total distance from all given strings is maximum (*farthest string problem*);
(iv) the consensus is a new string far from most of the given strings (*far from most string problem*).

Computational intractability of the general strings consensus problem was first proved in 1997 by Frances and Litman [7] and in 1999 by Sim and Park [28].

As a constring of the linear coding of DNA and proteins, many molecular biology problems have been formulated as computational optimization problems involving strings and sequences. Biological applications of computing distance/proximity among strings occur mainly in two varieties. Some require that a region of similarity be discovered, while other applications use the reverse complement of the region, such as designing probes or primers. In the following, some relevant biological applications are outlined.

Creating Diagnostic Probes for Bacterial Infections

Probes are strands of either DNA or RNA that have been modified (i.e., made either radioactive or fluorescent) so that their presence can be easily detected. One possible application of string problems arises in creating diagnostic probes for bacterial infections [14, 21]. In this scenario, given a set of DNA strings from a group of closely related pathogenic bacteria, the task is to find a substring that occurs in each of the bacterial strings (as close as possible) without occurring in the host's DNA. Probes are then designed to hybridize to these target strings, so that the detection of their presence indicates that at least one bacterial species is likely to be present in the host.

Primer Design

Primers are short strings of nucleotides designed such that they hybridize to a given DNA string or to all of a given set of DNA strings with the aim of providing a starting point for DNA strand synthesis by polymerase chain reaction (PCR). The hybridization of primers depends on several conditions, including some thermodynamic rules, but it is largely influenced by the number of mismatching positions among the given strings, and this number should be as small as possible [9, 10, 17].

Discovering Potential Drug Targets

Another biological application of string selection and comparison problems is related to discovering potential drug targets. Given a set of strings of orthologous genes from a group of closely related pathogens and a host (such as a human, crop, or livestock), the goal is to find a string fragment that is more conserved in all or most of the pathogens strings but not as conserved in the host. Information encoded by this fragment can then be used for novel antibiotic development or to create a drug that harms several pathogens with minimal effect on the host. All these applications reduce to the task of finding a pattern that, with some error, occurs in one set of strings (closest string problem) and/or does not occur in another set (farthest string problem). The far from most string problem can help to identify a string fragment that distinguishes the pathogens from the host, so the potential exists to create a drug that harms several but not all pathogens [9, 10, 17].

Motif Search

A motif is a string that occurs *approximately preserved* as a substring in some/-several of the DNA strings of a given set. Approximately preserved means that the motif occurs with changes in at most t positions for a fixed nonnegative integer t. The importance of a motif lies in its characteristic of being a candidate for substrings of noncoding parts of the DNA string that have functions related to, e.g., gene expression [9, 10].

For most consensus problems, Hamming distance is used instead of other alternative measures, such as, for example, the editing distance. The biological reasons justifying this choice are very well described and motivated by Lanctot et al. in [14–16] and can be summarized claiming that the "edit distance is more suitable to measure the *amount* of change that has happened, whereas the Hamming distance is more suitable to measure the *effect* of that change."

The remainder of this chapter is organized as follows. The next section lists notation and definitions used throughout the paper. The following four sections are devoted to the closest string, the close to most string, the farthest string, and the far from most string problem, respectively. All these problems are mathematically formulated and their properties analyzed. The most popular solution techniques for them are surveyed, along with the computational results obtained and analyzed in the literature. The second last section reports computational results which demonstrate empirically the efficiency of the state-of-the-art algorithms. Concluding remarks and future directions are discussed in the last section.

Notation

Throughout this chapter, the following notation and definitions will be used:

- An *alphabet* $\Sigma = \{c_1, c_2, \ldots, c_k\}$ is a finite set of elements, called *characters*.
- $s^i = (s^i_1, s^i_2, \ldots, s^i_m)$ denotes a string of m characters (that is, of length m) over alphabet Σ, i.e., $s^i_j \in \Sigma,\ j = 1, 2, \ldots, m$.
- Given two strings s^i and s^l on Σ such that $|s^i| = |s^l|$, $d_H(s^i, s^l)$ denotes their Hamming distance and is given by

$$d_H(s^i, s^l) = \sum_{j=1}^{|s^i|} \Phi(s^i_j, s^l_j), \tag{1}$$

  where s^i_j and s^l_j denote the character at position j in string s^i and in string s^l, respectively, and $\Phi : \Sigma \times \Sigma \to \{0, 1\}$ is a predicate function such that

$$\Phi(a, b) = \begin{cases} 0, & \text{if } a = b; \\ 1, & \text{otherwise.} \end{cases}$$

- For all consensus problems, each string s of length m over Σ is a valid solution.

The Closest String Problem (CSP)

Given a finite set of strings Ω on Σ, the problem is to find a *center string* $s^* \in \Sigma^m$ such that the Hamming distance between s^* and all strings in Ω is minimal; in other words, s^* is a string to which a minimal value d corresponds such that

$$d_H(s^*, s^i) \leq d, \quad \forall\, s^i \in \Omega.$$

The closest string problem can be formulated as an integer linear program (ILP). In fact, let $\Sigma_k \subseteq \Sigma$ be the set of characters appearing at position k in any of the strings from Ω.

For each $k = 1, 2, \ldots, m$ and $j \in V_k$, let us define the following binary variables:

$$x_{ck} = \begin{cases} 1, \text{ if character } c \in \Sigma_k \text{ is used at position } k \text{ of the solution;} \\ 0, \text{ otherwise.} \end{cases}$$

Then, the CSP admits the following ILP formulation:

$$\min d \tag{2}$$

subject to:

$$\sum_{c \in \Sigma_k} x_{ck} = 1 \quad \text{for } k = 1, 2 \ldots, m \tag{3}$$

$$m - \sum_{k=1}^{m} x_{s^i_k k} \leq d \quad \text{for } i = 1, 2 \ldots, n \tag{4}$$

$$d \in \mathbb{N}^+, \tag{5}$$

$$x_{ck} \in \{0, 1\}, \qquad \text{for } k = 1, 2, \ldots, m, \ \forall\, c \in \Sigma_k. \tag{6}$$

Equalities (3) guarantee that only one character is selected for each position $k \in \{1, 2, \ldots, m\}$. Inequalities (4) impose that if a character in a string s^i is not in the solution defined by the x-variables, then that character will contribute to increasing the Hamming distance from solution x to s^i. Finally, (5) forces d assume a nonnegative integer value and (6) define the decision variables.

This problem was first studied in the area of coding theory [27] and has been independently proved computationally intractable in [7, 15, 16].

In 2004, Meneses et al. [26] used a linear relaxation of the above-described mathematical model to design a branch and bound algorithm. At each iteration, the next node in the branching tree to be explored is the one with the smallest linear relaxation objective function value (also known as the *best-bound first strategy*). Once selected the next node and obtained an optimal fractional solution $x'_{ck} \in [0, 1]$, $k = 1, \ldots, m$, $\forall\, c \in \Sigma_k$, for the linear relaxation of the corresponding subproblem, the algorithm branches on the fractional variable x_{ck} with maximum value of x'_{ck}. The bounding phase is very important in any branch and bound algorithm: the better is the computed bound, the smaller is the number of nodes that need to be explored and that can therefore be pruned. For the bounding phase, Meneses et al. computed an initial bound selecting one of the given input strings and modifying it until a local optimal solution is found. The authors have empirically shown that their branch and bound algorithm is able to solve in reasonable running times small-size instances with 10–30 strings, each of which is 300–800 characters long.

Further exact methods proposed for the CSP are fixed-parameter algorithms [11, 20, 30] that are applicable only when the maximum Hamming distance among all pairs of strings is small. In fact, since these algorithms are designed to solve the decision version of the problem, it is necessary to apply them multiple times in order to find an optimal solution that minimizes the Hamming distance.

The first approximation algorithm for the CSP was proposed in [16] with a worst-case performance ratio of 2. It is a simple algorithm that constructs an approximate feasible solution in a pure random fashion. Starting from an empty solution, the algorithm selects at random the next element to be added to the solution under construction.

Better performance approximation algorithms proposed in the literature are based on the linear programming relaxation of the previously described ILP model. The basic idea consists in solving the linear programming relaxation of the ILP model and in using the result of the relaxed problem to find an approximate solution to the original problem. Following this line, [16] also proposed a $\frac{4}{3}(1+\epsilon)$-approximation algorithm (for any small $\epsilon > 0$) that uses the randomized rounding technique for obtaining an integer 0–1 solution from the continuous solution for the relaxed problem. The randomized rounding technique works by defining the value of a Boolean variable $x \in \{0,1\}$ to be $x = 1$ with a certain probability y, where y is the value of the continuous variable corresponding to x in the relaxation of the original integer programming problem. In 1999, Li et al. [17] used the rounding idea to design a polynomial-time approximation scheme (PTAS). A PTAS is a special type of approximation algorithm that, for each $\epsilon > 1$, yields a performance guarantee of ϵ in polynomial time. Thus, this can be viewed as a way of getting solutions with guaranteed performance, for any desired threshold greater than one. The PTAS proposed in [17] is also based on randomized rounding that here is refined to check results for a large (but polynomially bounded) number of subsets of indices. However, since a large number of iterations involve the solution of a linear relaxation of an ILP model, the algorithm becomes impractical for any instance with large strings. To efficiently deal with real-world scenarios and/or medium- to large-sized problem instances, several heuristic and metaheuristic algorithms have been proposed in the last few years.

In 2005, Liu et al. [18] designed a genetic algorithm and a simulated annealing algorithm, both in their sequential and their parallel versions. Genetic algorithms (GAs) are population-based metaheuristics that have been applied to find optimal or near-optimal solutions to combinatorial optimization problems [8, 12]. They implement the concept of *survival of the fittest* making an analogy between a solution and an *individual* in a *population*. Each individual has a corresponding *chromosome* that encodes the solution. A chromosome consists of a string of *genes*. Each gene can take on a value, called an *allele*, from some alphabet. A chromosome has an associated *fitness level* which is correlated to the corresponding objective function value of the solution it encodes. Over a number of iterations, called *generations*, GAs evolve a population of chromosomes. This is usually accomplished by simulating the process of natural selection through mating and mutation. For the CSP, starting from an initial randomly generated population, at each generation $0 \leq t \leq \texttt{number}-\texttt{generations}$ of the Liu et al.'s GA, a population $P(t)$ of `popsize` strings of length m is evolved, and the fitness function to be maximized is defined as the difference $m - d_{\max}$, where $d_{\max}$ is the largest Hamming distance between an individual of the population $P(t)$ and any string in Ω. The production of offspring in a GA is done through the process of mating or crossover, and Liu et al. used a multipoint crossover (MPX). In more detail, at a generic generation t, two parental individuals x and y in $P(t)$ are randomly chosen according to a probability which is proportional to their fitness. Then, iteratively until the offspring is not complete, x and y exchange parts between two randomly picked points. The resulting offspring have a new order of the strings, one part from the *first parent* and the other part from the *second parent*. Afterward, a mutation of any individual in the current population $P(t)$ is executed with some given probability. During this phase, two positions are randomly chosen and exchanged in the individual.

In their paper, Liu et al. proposed also a simulated annealing (SA) algorithm for the CSP. Originally proposed in [13], in the optimization and computer science research communities, simulated annealing is commonly said to be the "oldest" among the metaheuristics and surely one of the first techniques that had an explicit strategy to escape from local minima. Its fundamental idea is to allow moves resulting in solutions of worse quality in terms of objective function value than the current solution (uphill moves) in order to escape from local minima. The origin of simulated annealing and the choice of the acceptance criterion of a better quality solution lie in the physical annealing process that can be modeled by methods based on Monte Carlo techniques. One of the early Monte Carlo techniques for simulating the evolution of a solid in a heat bath to thermal equilibrium is due to [24], who in 1953 designed a method that iteratively (until a stopping criterion is met) generates a string of states of the solid in the following way. At a generic iteration k, given a current state i of the solid (i.e., a current solution x),

- E_i is the energy of the solid in state i (objective function value $f(x)$).
- a subsequent state j (solution $\overline{x}$) is generated with energy E_j (objective function value $f(\overline{x})$) by applying a *perturbation mechanism* such as displacement of a single particle ($\overline{x}$ is a solution "close" to x);
- if $E_j - E_i < 0$ (i.e., $\overline{x}$ is a better quality solution), j ($\overline{x}$) is accepted; otherwise, j ($\overline{x}$) is accepted with probability given by

$$\exp\left(-\frac{E_j - E_i}{k_B T_k}\right) \qquad \left[\exp\left(-\frac{f(\overline{x}) - f(x)}{k_B \cdot T_k}\right)\right],$$

where T_k is the heat bath temperature and k_B is the Boltzmann constant.

As the number of performed iterations increases, the current temperature T_k is decreased, resulting in a smaller probability of accepting not improving solutions. For the CSP, Liu et al.'s SA sets the initial temperature T_0 to $\frac{m}{2}$. The current temperature T_k is reduced every 100 iterations according to the "geometric cooling schedule" that is, $T_{k+100} = \gamma \cdot T_k$, where $\gamma = 0.9$. The stopping criterion is to reach a current temperature less than or equal to 0.001. Despite the interesting ideas proposed in [18], the experimental analysis involves only small instances with up to 40 strings of length 40.

For the special case of $|\Omega| = 3$ and $|\Sigma| = 2$, [19] designed an exact approach called distance first algorithm (DFA), whose basic idea is to let the Hamming distance $d_H(s^*, s^i)$, $i = 1, 2, 3$, be as close as possible. The algorithm decreases the distance between the string that is farthest to the other two strings and solution s^* while increasing the distance between the string that is closest to the other two strings and solution s^*. For the general case, the authors proposed a polynomial-time heuristic resulting from a combination of local search strategies inspired by [26] and an approximation algorithm called Largest Distance Decreasing Algorithm (LDDA) which is based on similar ideas as DFA.

More recently, Tanaka [29] proposed a novel heuristic (TA) based on the Lagrangian relaxation of the ILP model of the problem that allows to decompose the problem into subproblems, each corresponding to a position of the strings. The proposed algorithm combines a Lagrangian multiplier adjustment procedure to obtain feasibility and a tabu search as local improvement procedure. In [4], Della Croce and Salassa described three relaxation-based procedures. One procedure (RA) rounds up the result of continuous relaxation, while the other two approaches (BCPA and ECPA) fix a subset of the integer variables in the continuous solution at the current value and let the solver run on the remaining (sub)problem. The authors also observed that all relaxation-based algorithms have been tested on *rectangular instances*, i.e., with $n \ll m$, and that the instances such that $n \geq m$ are harder to be solved due to the higher number of constraints imposed by the strings, which enlarges the portion of non-integer components in the continuous solution of the problem. In the attempt to overcoming this drawback, Croce and Garraffa [3] designed a multistart relaxation-based algorithm (called the selective fixing algorithm) that for a predetermined number of iterations takes a feasible solution as input and iteratively selects variables to be fixed at their initial value until the number of free variables is small enough that the remaining subproblem can be efficiently solved to optimality by an ILP solver. The new solution found by the solver can then be used as initial solution for the next iteration. The authors have experimentally shown that their algorithm is much more robust compared to the state-of-the-art competitors and is able to solve a wider set of instances of different types, including those with $n \geq m$.

The Close to Most String Problem (CTMSP)

The closest string problem can be seen as a special case of the so-called close to most string problem (CTMSP) that consists in determining a string close to most of the strings in the input set Ω. This can be formalized by saying that, given a threshold t, a string s^* must be found maximizing the variable l such that

$$d_H(s^*, s^i) \leq t, \text{ for } s^i \in P \subseteq \Sigma \text{ and } |P| = l.$$

$$\mathbf{max} \sum_{i=1}^{n} y_i \tag{7}$$

subject to:

$$\sum_{c \in \Sigma} x_{ck} = 1 \quad \text{for } k = 1, \ldots, m \tag{8}$$

$$\sum_{k=1}^{m} x_{s_k^i k} \geq m \cdot y_i - t \quad \text{for } i = 1, \ldots, n \tag{9}$$

$$x_{ck}, y_i \in \{0, 1\}$$

Constraints (8) ensure that for each position k of a possible solution, exactly one character from Σ_k is chosen. Constraints (9) ensure that y_i can only be set to 1 if and only if the number of differences between $s^i \in \Omega$ and the possible solution (as defined by the setting of the variables x_{ck}) is less than or equal to t. Remember, in this context, s_k^i denotes the character at position k in $s^i \in \Omega$.

Despite its similarity with the CSP, the CTMSP has not been widely studied. In [2] it was proved that this problem has no polynomial-time approximation scheme (**PTAS**) unless **NP** has randomized polynomial-time algorithms.

The Farthest String Problem (FSP)

Given a finite set of strings Ω over alphabet Σ, a problem complementary to the CSP is that of finding a string $s^* \in \Sigma^m$ farthest from the strings in Ω. This type of problem can be useful in situations such as finding a genetic string that cannot be associated to a given number of species.

Like the CSP, the farthest string problem can be formulated mathematically in form of an ILP, where both decision variables and constraints have an interpretation which is contrary to the one in the CSP:

$$\max d \tag{10}$$

subject to:

$$\sum_{c \in \Sigma_k} x_{ck} = 1 \quad \text{for } k = 1, 2 \ldots, m \tag{11}$$

$$m - \sum_{k=1}^{m} x_{s_k^i k} \geq d \quad \text{for } i = 1, 2 \ldots, n \tag{12}$$

$$d \in \mathbb{N}^+, \tag{13}$$

$$x_{ck} \in \{0, 1\}, \qquad \text{for } k = 1, 2, \ldots, m, \ \forall\, c \in \Sigma_k. \tag{14}$$

The computational intractability of the FSP was demonstrated in [16], where it was proved that the problem remains intractable even for the simplest case where the alphabet has only two characters.

Despite its inherent computational intractability, it has be shown in [16] that there is a PTAS for the FSP. The algorithm is based on the randomized rounding of the relaxed solution of the above-reported ILP and uses the randomized rounding technique together with probabilistic inequalities to determine the maximum error possible in the solution computed by the algorithm. Note that, the mathematical formulation of FSP is quite similar to the one used for the CSP, with only a change in the optimization objective, and the inequality sign in the constraint

$$m - \sum_{j=1}^{m} x_{s^i_j j} \geq d, \qquad i = 1, 2, \ldots, n.$$

Thus, to solve the problem using an ILP formulation, one can use similar techniques to those employed for solving the CSP. In 2011 [31] and more recently in 2015 Zörnig [32] proposed a few integer programming models for some variants of the farthest string problem and the closest string problem. The number of variables and constraints is substantially less compared with state-of-the-art integer linear programming models, and the solution of the linear programming relaxation contains only a small proportion of non-integer values, which considerably simplifies both a subsequent rounding process and a branch and bound procedure.

The Far From Most String Problem (FFMSP)

A problem closely related to the farthest string problem is the far from most string problem (FFMSP). It consists in determining a string far from most of the strings in the input set Ω. This can be formalized by saying that, given a threshold t, a string s^* must be found maximizing the variable l such that

$$d_H(s^*, s^i) \geq t, \text{ for } s^i \in P \subseteq \Sigma \text{ and } |P| = l.$$

In [1], the FFMSP has been mathematically formulated as an ILP. In fact, by defining a Boolean variable x_{ck} for each position k ($k = 1, 2 \ldots, m$) of a possible solution and for each character $c \in \Sigma_k$ and a Boolean variable y_i ($i = 1, 2 \ldots, n$) for each of the n input strings provided in set Ω, the FFMSP can be stated as the following ILP:

$$\max \sum_{i=1}^{n} y_i \tag{15}$$

subject to:

$$\sum_{c \in \Sigma_k} x_{ck} = 1 \quad \text{for } k = 1, 2 \ldots, m \tag{16}$$

$$\sum_{k=1}^{m} x_{s^i_k k} \leq m - t \cdot y_i \quad \text{for } i = 1, 2 \ldots, n \tag{17}$$

$$x_{ck}, y_i \in \{0, 1\}$$

Constraints (16) ensure that for each position k of a possible solution, exactly one character from Σ_k is chosen. Constraints (17) ensure that y_i can only be set to 1 if and only if the number of differences between $s^i \in \Omega$ and the possible solution (as defined by the setting of the variables x_{ck}) is greater than or equal to t. Remember, in this context, s^i_k denotes the character at position k in $s^i \in \Omega$.

Despite the similarity, it can be shown [16] that the FFMSP is much harder to approximate than the FSP, due to the approximation preserving reduction to FFMSP from the independent set problem, a classical and computationally intractable combinatorial optimization problem. In particular, [16] demonstrated that for strings over an alphabet Σ with $|\Sigma| \geq 3$, approximating the FFMSP within a polynomial factor is **NP**-hard.

The first attempt in the direction of the design of heuristic methods to efficiently solve the FFMSP was done in [22, 23], who proposed a heuristic algorithm consisting of a simple greedy construction followed by an iterative improvement phase. Later, [6] designed a simple GRASP, recently improved in [25]. Mousavi et al. noticed that the search landscape of the FFMSP is characterized by many solutions having the same objective value. Consequently, local search is likely to visit many suboptimal local maxima. To efficiently escape from these local maxima, Mousavi et al. devised a new hybrid heuristic evaluation function and used it in conjunction with the objective function when evaluating neighbor solutions during the local search phase in the GRASP framework.

Ferone et al. [5] designed the following pure and hybrid multistart iterative heuristics:

- a pure GRASP, inspired by [6];
- a GRASP that uses forward path-relinking for intensification;
- a pure VNS;
- a VNS that uses forward path-relinking for intensification;
- a GRASP that uses VNS to implement the local search phase; and
- a GRASP that uses VNS to implement the local search phase and forward path-relinking for intensification.

The algorithms were tested on several random instances, and the results showed that the hybrid GRASP with VNS and forward path-relinking always found much better quality solutions compared with the other algorithms, but clearly with higher running times as compared to the pure GRASP and the hybrid GRASP with forward path-relinking. The best objective function values found by GRASP and its hybrids were when the construction phase was more greedy than random. The integration of forward path-relinking as an intensification procedure in the pure metaheuristics was beneficial in terms of solution quality. A further investigation conducted, studying the empirical distributions of the random variable *time-to-target-solution value*, revealed that, given any fixed amount of computing time, GRASP with forward path-relinking has an empirically higher probability than all competitors of finding a target solution.

In [1], besides the first linear integer programming formulation for the FFMSP described above (15), (16), and (17), a hybrid ant colony optimization approach has been proposed. This hybrid approach consists of two phases. A first phase applies ant colony optimization until the convergence of the pheromone values is reached. After this first phase, the algorithm possibly applies a second phase in which the hybridization with a mathematical programming solver takes place. Both the linear integer programming formulation and the hybrid ant colony algorithm have compared to the most performing hybrid GRASP with path-relinking, and computational results on a large set of randomly generated test instances have indicated that the hybrid ACO is very competitive.

A Simple ILP-Based Heuristic

In the following we present the results of a quite simple ILP-based heuristic which can be applied to all four problems described before: the closest string problem (CSP), the clost to most string problem (CTMSP), the farthest string problem (FSP), and the far from most string problem (FFMSP). The heuristic is based on the ILP models of the four problems. It works as follows. Given a fixed computation time limit (t_{limit}), maximally half of this computation time is given to an ILP solver for tackling the mixed integer linear problem (MILP) that is obtained by relaxing the x_{ck}-variables involved in all four ILP models. The used ILP solver returns the best solution found in the given computation time. Note that this solution may, or may not, correspond to the optimal MILP solution. The fractional values of the x_{ck}-variables after termination of the solver are henceforth denoted by x'_{ck}. In the second phase of the heuristic, the corresponding ILP models are solved in the remaining computation time, with the following additional constraints:

$$x_{ck} = 1 \quad \text{for } k = 1, \ldots, m, \; c \in \Sigma_k, \; x'_{ck} = 1 \tag{18}$$

In other words, whenever a variable x_{ck} in the best-found MILP solution has a value of 1, this value is fixed for the solution of the ILP model.

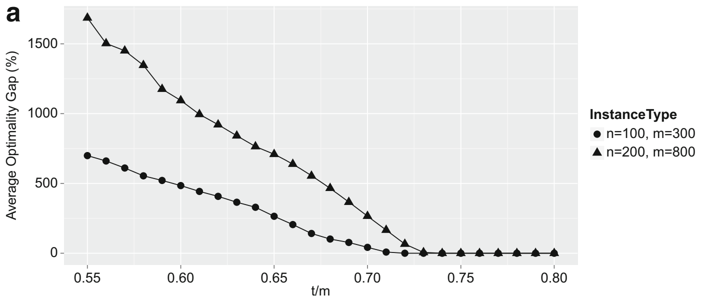
Experimental Evaluation

The heuristic described above is compared, in the case of all four problems, with the application of the ILP solver to the original ILP models. Moreover, a simple greedy algorithm was applied. This greedy algorithm generates a solution by selecting for each position of the solution string the letter which, in the case of the CSP and the CTMSP, has the least number of appearances at this position in the set of input strings and which, in the case of the FSP and the FFMSP, has the highest number of appearances at this position in the set of input strings. The greedy algorithm is henceforth denoted by GREEDY. As (M)ILP solver we used IBM ILOG CPLEX V12.1. The experimental results were obtained on a cluster of PCs with "Intel(R) Xeon(R) CPU 5160" CPUs of four nuclei of 3000 MHz and 4 GB of RAM. Moreover, CPLEX was configured for single-threaded execution. Depending on the problems, we used two different computation time limits: $t_{\text{limit}} = 200$ s and $t_{\text{limit}} = 3600$ s per problem instance. The different applications of CPLEX and the ILP-based heuristic, respectively, are named accordingly: CPLEX-200, CPLEX-3600, HEURISTIC-200, and HEURISTIC-3600.

All the algorithms described above were applied, in the context of all four problems, to a set of benchmark instances that was originally introduced in [5] for the FFMSP. This set consists of random instances of different size. More specifically, the number of input strings (n) is in $\{100, 200\}$, and the length of the input strings (m) is in $\{300, 600, 800\}$. In all cases, the alphabet size is four, that is, $|\Sigma| = 4$. For each combination of n and m, the set consists of 100 random instances. This makes a total of 600 instances. Note that, in the context of the CTMSP and the FFMSP, a value for parameter t must be specified before running the algorithm(s). However, a sensible choice of t is not trivial. For example, in the context of the CTMSP, the lower the value of t, the easier it should be, for example, for CPLEX to solve the problem to optimality. In order to be able to choose meaningful values for t, the following experiments were executed. CPLEX was applied to each problem instance—both concerning the CTMSP and the FFMSP—for each value of $t \in \{0.05, 0.02, \ldots, 0.95\,\text{m}\}$. This was done with a time limit of 3600 s per run. The corresponding optimality gaps (averaged over 100 problem instances) and the number of instances (out of 100) that was solved to optimality are graphically presented in Fig. 1. The results reveal that, in the case of the CTMSP, the problem becomes difficult for approx. $t \leq 0.72$ m. In the case of the FFMSP, the problem becomes difficult for approx. $t \geq 0.78$ m. Therefore, the following values for t were chosen for the final experimental evaluation: $t \in \{0.65, 0.7, 0.75\,\text{m}\}$ in the case of the CTMSP and $t \in \{0.75, 0.8, 0.85\,\text{m}\}$ in the case of the FFMSP.

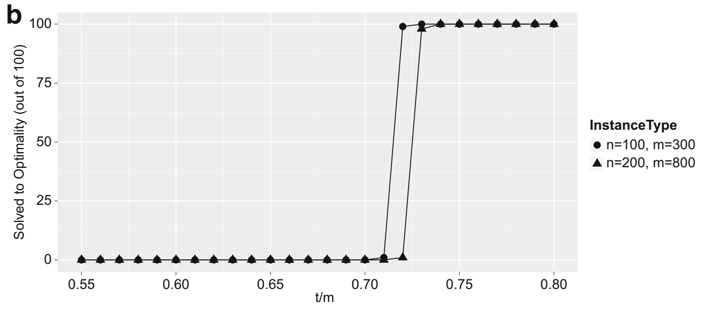
The numerical results for the CSP are shown in Table 1. They are presented as averages over the 100 instances for each combination of n (the number of input strings) and m (the length of the input strings). For all three algorithms, we provide the values of the best-found solutions (averaged over 100 problem instances) and the computation time at which these solutions were found. In the case of CPLEX-200, the average optimality gap is additionally provided. The results clearly show that HEURISTIC-200 outperforms both GREEDY and CPLEX-200.

a



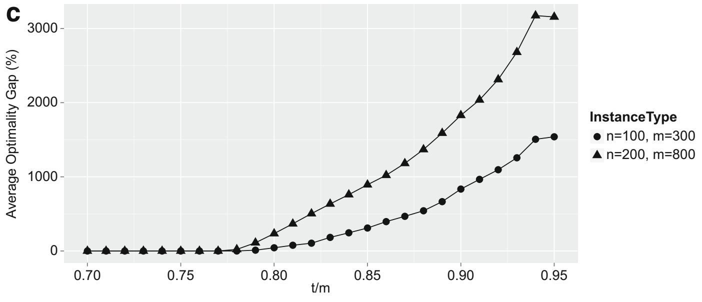
Average optimality gap (in percent) of CPLEX for the CTMSP.

b



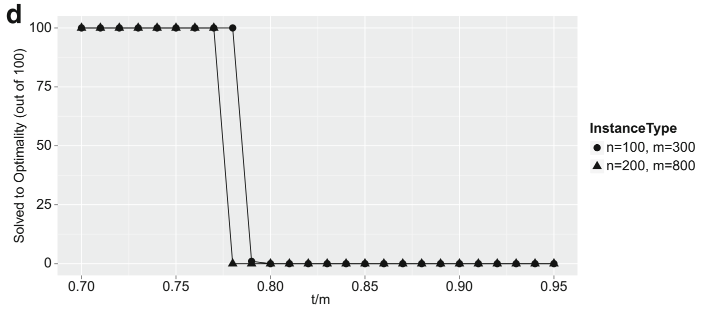
Number of instances solved (out of 100) by CPLEX for the CTMSP.

c



Average optimality gap (in percent) of CPLEX for the FFMSP.

d



Number of instances solved (out of 100) by CPLEX for the FFMSP.

Fig. 1 Justification for the choice of parameter t in the context of the CTMSP and the FFMSP. (**a**) Average optimality gap (in percent) of CPLEX for the CTMSP. (**b**) Number of instances solved (out of 100) by CPLEX for the CTMSP. (**c**) Average optimality gap (in percent) of CPLEX for the FFMSP. (**d**) Number of instances solved (out of 100) by CPLEX for the FFMSP

Table 1 Numerical results for the CSP

		Greedy		Heuristic-200		Cplex-200		
n	m	Value	Time	Value	Time	Value	Time	Gap
100	300	228.55	<0.009	**213.33**	10.79	213.48	10.53	0.88
100	600	446.91	<0.009	**422.90**	7.36	423.06	5.22	0.47
100	800	590.08	<0.009	**562.40**	6.86	562.52	7.19	0.36
200	300	235.02	<0.009	**219.99**	12.74	220.03	33.42	1.38
200	600	457.19	<0.009	**434.33**	14.03	434.36	67.29	0.78
200	800	604.49	<0.009	**576.95**	11.59	577.01	87.03	0.60

Table 2 Numerical results for the FSP

		Greedy		Heuristic-200		Cplex-200		
n	m	Value	Time	Value	Time	Value	Time	Gap
100	300	221.67	<0.009	**236.53**	4.46	236.41	9.34	0.68
100	600	455.15	<0.009	**476.55**	7.13	476.39	7.10	0.38
100	800	611.61	<0.009	**636.58**	10.13	636.45	9.09	0.27
200	300	215.74	<0.009	230.11	9.29	**230.14**	25.72	1.24
200	600	443.88	<0.009	**465.68**	10.81	465.59	53.41	0.67
200	800	596.41	<0.009	**622.81**	10.52	622.72	60.49	0.51

Similar conclusions can be drawn in the case of the FSP, for which the results are presented in Table 2. Except for one case ($n = 200$, $m = 300$), Heuristic-200 outperforms both Greedy and Cplex-200.

In the case of the CTMSP, both the ILP-based heuristic and CPLEX were applied with computation time limits 200 and 3600 CPU seconds. Therefore, Table 3 contains results for Heuristic-200, Heuristic-3600, Cplex-200, and Cplex-3600. The following observations can be made:

- Both CPLEX and the ILP-based heuristic greatly outperform Greedy.
- When the problem is rather easy—that is, for a setting of $t = 0.75\,\text{m}$—CPLEX has usually slight advantages over the ILP-based heuristic. This is the case especially for the instances with larger number of input strings ($n = 200$).
- With growing problem difficulty, the ILP-based heuristic starts to outperform CPLEX. In particular, for a setting of $t = 0.65\,\text{m}$, the differences in the qualities of the obtained solutions between the ILP-based heuristic and CPLEX are significant.

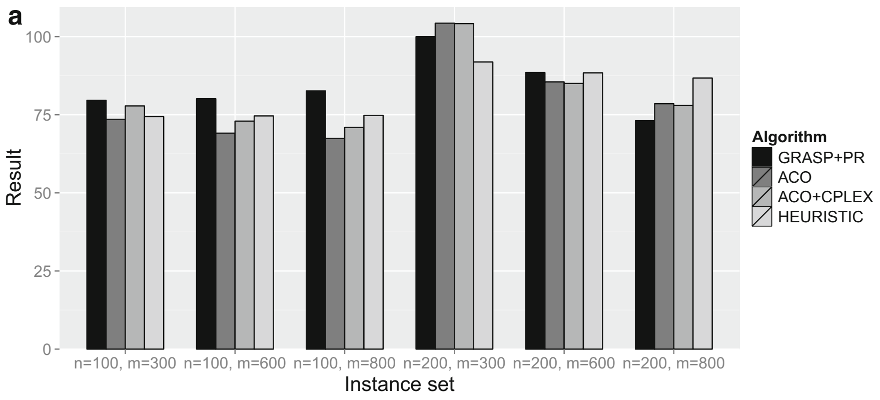
Not surprisingly, the same observations can be made in the context of the FFMSP, for which the results are provided in Table 4. As the considered benchmark instances were originally used for the FFMSP, we are able to compare to current state-of-the-art results for this problem (see [1]). This comparison is graphically presented in Fig. 2 for all instances concerning the interesting cases $t = 0.8\,\text{m}$ and $t = 0.85\,\text{m}$. The results show that, for $t = 0.8\,\text{m}$, Heuristic-3600 is generally outperformed by the other state-of-the-art methods. However, note that when the instance size

Table 3 Numerical results for the CTMS problem

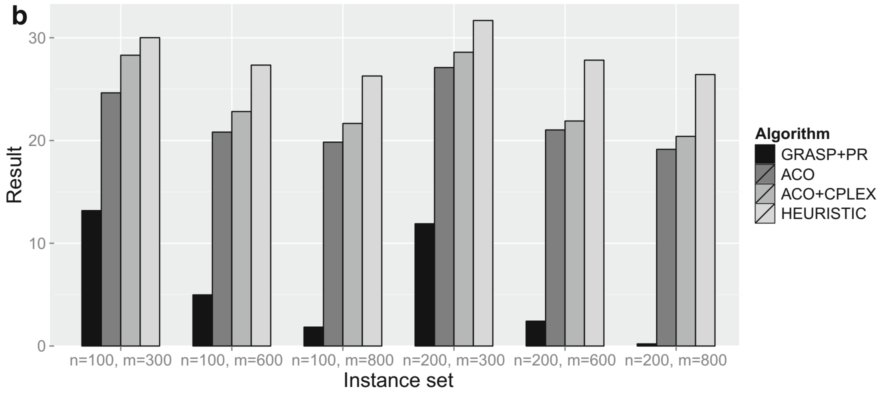
			GREEDY		HEURISTIC-200		HEURISTIC-3600		CPLEX-200			CPLEX-3600		
n	m	t	Value	Time	Value	Time	Value	Time	Value	Time	Gap	Value	Time	Gap
100	300	0.65 m	4.18	<0.009	31.87	140.45	**33.79**	2315.36	23.39	110.15	262.64 %	27.14	1509.07	210.56 %
100	300	0.7 m	55.94	<0.009	72.08	125.04	**74.75**	1642.93	69.95	62.70	41.98 %	72.14	1411.02	37.19 %
100	300	0.75 m	98.30	<0.009	99.98	2.55	**100.00**	4.45	**100.00**	0.81	0.00 %	**100.00**	1.06	0.00 %
100	600	0.65 m	0.70	<0.009	29.28	130.18	**31.45**	2182.52	21.96	100.77	289.23 %	23.16	1165.62	268.94 %
100	600	0.7 m	56.05	<0.009	72.50	77.33	**75.05**	876.32	71.30	109.45	40.03 %	73.44	1106.52	35.80 %
100	600	0.75 m	99.74	<0.009	**100.00**	1.17	**100.00**	1.20	**100.00**	1.72	0.00 %	**100.00**	1.97	0.00 %
100	800	0.65 m	0.18	<0.009	28.13	126.89	**30.35**	2131.77	20.14	70.47	328.33 %	25.86	1866.29	228.77 %
100	800	0.7 m	56.84	<0.009	73.14	51.06	**76.03**	837.32	72.88	119.22	37.34 %	75.23	1052.56	32.93 %
100	800	0.75 m	99.94	<0.009	99.99	2.24	**100.00**	2.91	**100.00**	2.31	0.00 %	**100.00**	2.16	0.00 %
200	300	0.65 m	2.16	<0.009	32.12	146.42	**36.93**	2493.67	23.81	94.62	576.99 %	24.30	735.91	562.71 %
200	300	0.7 m	65.51	<0.009	81.25	145.30	**92.82**	2720.97	85.61	144.05	121.06 %	92.37	1784.17	103.87 %
200	300	0.75 m	186.07	<0.009	192.90	43.00	195.05	595.86	**200.00**	3.22	0.00 %	**200. 00**	3.05	0.00 %
200	600	0.65 m	0.05	<0.009	28.46	129.11	**32.54**	2242.65	20.01	32.28	712.81 %	23.25	1192.03	594.35 %
200	600	0.7 m	50.04	<0.009	70.71	137.05	**88.04**	2570.76	66.95	118.40	185.15 %	81.96	1794.91	132.00 %
200	600	0.75 m	196.29	<0.009	196.23	44.52	198.05	647.22	**200.00**	4.86	0.00 %	**200. 00**	4.75	0.00 %
200	800	0.65 m	0.03	<0.009	27.21	128.22	**30.84**	2141.78	20.13	46.58	709.14 %	22.70	2238.87	615.48 %
200	800	0.7 m	43.27	<0.009	67.18	137.80	**86.59**	2552.02	63.94	122.52	198.62 %	79.93	2266.09	138.14 %
200	800	0.75 m	198.40	<0.009	196.50	44.65	198.27	623.50	**200.00**	7.00	0.00 %	**200. 00**	9.00	0.00 %

Table 4 Results for the FFMS problem

			GREEDY		HEURISTIC-200		HEURISTIC-3600		CPLEX-200			CPLEX-3600		
n	m	t	Value	Time	Value	Time	Value	Time	Value	Time	Gap	Value	Time	Gap
100	300	0.75 m	98.40	<0.009	**100.00**	0.09	**100.00**	0.16	**100.00**	0.10	0.00 %	**100.00**	0.11	0.00 %
100	300	0.8 m	54.67	<0.009	71.79	134.70	**74.41**	2594.31	69.87	69.91	42.69 %	71.56	1069.58	38.61 %
100	300	0.85 m	1.71	<0.009	28.83	118.05	**30.01**	1831.47	23.41	118.42	298.96 %	26.37	1881.75	249.42 %
100	600	0.75 m	99.82	<0.009	**100.00**	0.29	**100.00**	0.31	**100.00**	0.30	0.00 %	**100.00**	0.34	0.00 %
100	600	0.8 m	53.52	<0.009	71.88	113.65	**74.64**	1976.14	70.93	81.89	40.91 %	71.97	575.00	38.64 %
100	600	0.85 m	0.10	<0.009	25.93	107.92	**27.34**	1808.34	20.37	101.20	360.32 %	24.11	2014.55	285.93 %
100	800	0.75 m	99.99	<0.009	**100.00**	0.49	**100.00**	0.53	**100.00**	0.47	0.00 %	**100.00**	0.56	0.00 %
100	800	0.8 m	53.42	<0.009	71.88	111.76	**74.79**	1685.50	71.17	105.51	40.63 %	72.55	514.75	37.80 %
100	800	0.85 m	0.04	<0.009	25.08	105.12	**26.28**	1804.13	19.82	109.72	374.61 %	22.46	1573.26	317.03 %
200	300	0.75 m	188.58	<0.009	199.99	0.35	199.99	0.36	**200.00**	0.33	0.00 %	**200.00**	0.43	0.00 %
200	300	0.8 m	60.63	<0.009	78.58	124.40	**91.91**	2635.07	82.90	143.63	136.67 %	89.56	1775.12	118.20 %
200	300	0.85 m	0.61	<0.009	28.41	113.99	**31.68**	1869.94	20.13	71.95	813.23 %	24.88	2155.67	635.34 %
200	600	0.75 m	196.81	<0.009	**200.00**	0.56	**200.00**	0.60	**200.00**	0.57	0.00 %	**200.00**	0.68	0.00 %
200	600	0.8 m	43.29	<0.009	69.11	138.76	**88.42**	2459.95	62.67	116.46	213.91 %	77.34	1778.80	153.80 %
200	600	0.85 m	0.00	<0.009	24.93	103.93	**27.82**	1815.27	19.88	128.44	826.66 %	21.53	1173.71	753.46 %
200	800	0.75 m	198.75	<0.009	**200.00**	0.93	**200.00**	0.94	**200.00**	0.93	0.00 %	**200.00**	1.11	0.00 %
200	800	0.8 m	36.03	<0.009	69.45	139.97	**86.77**	2536.96	59.55	110.48	230.10 %	73.93	2565.90	166.09 %
200	800	0.85 m	0.00	<0.009	23.82	103.74	**26.42**	1803.11	18.67	47.26	887.78 %	20.79	808.43	784.97 %



Results for all instances with $t = 0.8m$.



Results for all instances with $t = 0.85m$.

Fig. 2 Graphical representation of the comparison between HEURISTIC-3600 and the current FFMSP state-of-the-art methods (GRASP+PR, ACO, and ACO+CPLEX) for $t = 0.8\,\text{m}$ (see (**a**)) and $t = 0.85\,\text{m}$ (see (**b**))

(in terms of the number of input strings and their length) grows, HEURISTIC-3600 starts to produce better results than the competitors. In the case of $t = 0.85\,\text{m}$, which results in more difficult instances than $t = 0.8\,\text{m}$, HEURISTIC-3600 clearly outperforms the current state-of-the-art methods.

Conclusions

The goal of this chapter was to provide an overview of some string selection and comparison problems, with special emphasis on the optimization and operational research perspective. Besides mathematical models that can be used to find exact solutions only up to a certain instance size, there are many approximate techniques, proposed by researchers from several heterogenous communities, which are more or less efficient, depending on the input data and the type of information they make use of. Not surprisingly, generally there is no single best approach that wins in every aspect. Therefore, we proposed a simple ILP-based heuristic that can be used for any of the four considered problems. We have shown that this heuristic outperforms both a general greedy algorithm and the application of an ILP solver (CPLEX) to the original ILP models. In the case of the far from most string problem, we were even able to show that this simple heuristic is able to produce state-of-the-art results for instances which are intrinsically difficult to be solved.

Cross-References

▶ Genetic Algorithms
▶ GRASP
▶ Variable Neighborhood Descent
▶ Variable Neighborhood Search

Acknowledgments C. Blum was supported by project TIN2012-37930 of the Spanish Government. In addition, support is acknowledged from IKERBASQUE (Basque Foundation for Science).

References

1. Blum C, Festa P (2014) A hybrid ant colony optimization algorithm for the far from most string problem. In: Proceedings of the 14th European conference on evolutionary computation in combinatorial optimisation (EvoCOP2014). Lecture notes in computer science, vol 8600. Springer, Berlin, pp 1–12
2. Boucher C, Landau G, Levy A, Pritchard D, Weimann O (2013) On approximating string selection problems with outliers. Theor Comput Sci 498:107–114
3. Croce FD, Garraffa M (2014) The selective fixing algorithm for the closest string problem. Comput Oper Res 41:24–30
4. Croce FD, Salassa F (2012) Improved lp-based algorithms for the closest string problem. Comput Oper Res 39:746–749
5. Ferone D, Festa P, Resende M (2013) Hybrid metaheuristics for the far from most string problem. In: Proceedings of 8th international workshop on hybrid metaheuristics. Lecture notes in computer science, vol 7919. Springer, Berlin, pp 174–188
6. Festa P (2007) On some optimization problems in molecular biology. Math Biosci 207(2): 219–234
7. Frances M, Litman A (1997) On covering problems of codes. Theory Comput Syst 30(2): 113–119
8. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Reading, Massachusetts, Addison-Wesley
9. Gramm J (2003) Fixed-parameter algorithms for the consensus analysis of genomic data. Eberhard Karls University of Tübingen, PhD thesis
10. Gramm J, Hüffner F, Niedermeier R (2002) Closest strings, primer design, and motif search. In: Sixth annual international conference on computational molecular biology, Washington, DC, pp 74–75

11. Gramm J, Niedermeier R, Rossmanith P (2003) Fixed-parameter algorithms for closest string and related problems. Algorithmica 37:25–42
12. Holland JH (1975) Adaptation in natural and artificial systems. Cambridge, Massachusetts, MIT Press
13. Kirkpatrick S (1984) Optimization by simulated annealing: quantitative studies. J Stat Phys 34(5–6):975–986
14. Lanctot J (2004) Some string problems in computational biology. University of Waterloo, PhD thesis
15. Lanctot J, Li M, Ma B, Wang S, Zhang L (1999) Distinguishing string selection problems. In: Proceedings of the annual ACM-SIAM symposium on discrete aLgorithms (SODA), Baltimore, pp 633–642
16. Lanctot J, Li M, Ma B, Wang S, Zhang L (2003) Distinguishing string selection problems. Inf Comput 185(1):41–55
17. Li M, Ma B, Wang L (1999) Finding similar regions in many strings. In: ACM symposium on theory of computing (STOC'99), New York, pp 473–482
18. Liu X, He H, Sýkora O (2005) Parallel genetic algorithm and parallel simulated annealing algorithm for the closest string problem. In: Proceedings of ADMA 2005. Lecture notes in artificial intelligence, vol 3584. Springer, Berlin Heidelberg New York, pp 591–597
19. Liu X, Liu S, Hao Z, Mauch H (2011) Exact algorithm and heuristic for the closest string problem. Comput Oper Res 38(11):1513–1520
20. Ma B, Sun X (2008) More efficient algorithms for closest string and substring problems. Lecture notes in computer science, vol 4955. Springer, Berlin, pp 396–409
21. Macario A, de Macario EC (eds) (1990) Gene probes for bacteria. Academic Press, San Diego
22. Meneses C, Oliveira C, Pardalos P (2005) Optimization techniques for string selection and comparison problems in genomics. IEEE Eng Med Biol Mag 24(3):81–87
23. Meneses C, Pardalos P, Resende M, Vazacopoulos A (2005) Modeling and solving string selection problems. In: Proceedings of the 2005 international symposium on mathematical and computational biology (BIOMAT 2005), pp 55–65
24. Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092
25. Mousavi S, Babaie M, Montazerian M (2012) An improved heuristic for the far from most strings problem. J Heuristics 18:239–262
26. Pardalos P, Oliveira C, Lu Z, Meneses C (2004) Optimal solutions for the closest string problem via integer programming. INFORMS J Comput 16:419–429
27. Roman S (1992) Coding and information theory. Graduate texts in mathematics, vol 134. Springer, New York
28. Sim J, Park K (1999) The consensus string problem for a metric is NP-complete. In: Proceedings of the annual Australasian workshop on combinatorial algorithms (AWOCA), pp 107–113
29. Tanaka S (2012) A heuristic algorithm based on Lagrangian relaxation for the closest string problem. Comput Oper Res 39:709–717
30. Wang L, Zhu B (2009) Efficient algorithms for the closest string and distinguishing string selection problems. Lecture notes in computer science, vol 5598. Springer, Berlin, pp 261–270
31. Zörnig P (2011) Improved optimization modelling for the closest string and related problems. Appl Math Modell 35(12):5609–5617
32. Zörnig P (2015) Reduced-size integer linear programming models for string selection problems: application to the farthest string problem. J Comput Biol 22(8):729–742