A Hybrid Ant Colony Optimization Algorithm for the Far From Most String Problem

Christian Blum[1,2] and Paola Festa[3]

[1] Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain
christian.blum@ehu.es
[2] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
[3] Department of Mathematics and Applications "R. Caccioppoli", University of Napoli FEDERICO II, Italy
paola.festa@unina.it

Abstract. The *far from most string problem* belongs to the family of string selection and comparison problems known as *sequence consensus problems*, where a finite set of sequences is given and one is interested in finding their consensus, that is, a new sequence that represents as much as possible all the given sequences. Among the consensus problems, the far from most string problem is computationally one of the hardest ones with applications in several fields, including molecular biology where one is interested in creating diagnostic probes for bacterial infections or in discovering potential drug targets.

This paper comes with several contributions. On one side, the first linear integer programming formulation for the considered problem is introduced. On the other side, a hybrid ant colony optimization approach for finding good approximate solution to the problem is proposed. Both approaches are compared to the current state of the art, which is a recently proposed hybrid GRASP with path-relinking. Computational results on a large set of randomly generated test instances indicate that the hybrid ACO is very competitive.

1 Introduction

The combinatorial optimization problem tackled in this paper is known as the *far from most string problem* (FFMSP). It belongs to a family of string problems labelled as *sequence consensus problems*, where a finite set of sequences is given and one is interested in finding their consensus, that is, a new sequence that represents as much as possible all the given sequences. Several different (and even opposing) objectives may be considered in the context of sequence consensus problems. Examples include the following ones.

1. The consensus is a new sequence whose total distance from all given sequences is minimal (*closest string problem*) (CSP);
2. The consensus is a new sequence whose total distance from all given sequences is maximal (*farthest string problem*) (FSP);

C. Blum and G. Ochoa (Eds.): EvoCOP 2014, LNCS 8600, pp. 1–12, 2014.

3. The consensus is a new sequence far from most of the given sequences, which is the case of the FFMSP.

Apart from introducing a linear integer programming model, the main contribution of the paper is a hybrid ant colony optimization algorithm (ACO) [2] for tackling the FFMSP. ACO algorithms are metaheuristics inspired by the shortest path finding behaviour of natural ant colonies. The hybrid algorithm outlined in this work concerns a sequential combination of ant colony optimization with a mathematical programming solver. In the following, we first provide the necessary notation for being able to state the FFMSP in a technical way. After providing the problem definition we give a short summary on related work.

1.1 Notation

In order to be able to formally state the FFMSP, the following notation is introduced.

- An alphabet $\Sigma = \{c_1, \ldots, c_k\}$ is a finite set of k different characters.
- $s^i = (s^i_1, s^i_2, \ldots, s^i_m)$ denotes a sequence of m characters (that is, of length m) over alphabet Σ.
- Given two sequences s^i and s^l over Σ such that $|s^i| = |s^l|$, the (generalized) Hamming distance $d_H(s^i, s^l)$ between s^i and s^l is calculated as follows:

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

  where s^i_j and s^l_j denote the character at position j in sequence s^i and in sequence s^l, respectively, and $\phi : \Sigma \times \Sigma \mapsto \{0, 1\}$ is a predicate function such that $\phi(a, b) = 0$, if $a = b$, and $\phi(a, b) = 1$, otherwise.
- Given a set $\Omega = \{s^1, \ldots, s^n\}$ of n sequences of length m over Σ, d^Ω_H denotes the minimum Hamming distance between all sequences in Ω. Formally,

$$d^\Omega_H := \min \left\{ d_H(s^i, s^l) \mid i, l \in \{1, \ldots, n\}, i < l \right\} \ . \tag{2}$$

  Note that $0 \leq d^\Omega_H \leq m$.
- Each sequence s of length m over Σ is a valid solution to the FFMSP. Given any valid solution s and a threshold value $0 \leq t \leq m$, set $P^s \subseteq \Omega$ is defined as follows:

$$P^s := \{s^i \in \Omega \mid d_H(s^i, s) \geq t\}. \tag{3}$$

1.2 Problem Definition

Given a fixed threshold value t (where $0 \leq t \leq m$), a finite alphabet Σ of size k, and a set $\Omega = \{s^1, \ldots, s^n\}$ of n sequences of length m over Σ, the goal of the FFMSP consists in finding a sequence s^* of length m over Σ such that P^{s^*}

is of maximal size. In other words, given a solution s, the objective function $f: \Sigma^m \mapsto \{1, \ldots, n\}$ is defined as follows:

$$f(s) := |P^s| \tag{4}$$

Among the sequence consensus problems, the FFMSP is one of the computationally hardest. In fact, compared to the other consensus problems, it is much harder to approximate, due to the approximation preserving reduction to the FFMSP from the independent set problem, which is a classical and computationally intractable combinatorial optimization problem. In 2003, Lanctot et al. [11] proved that for sequences over an alphabet Σ with $|\Sigma| \geq 3$, approximating the FFMSP within a polynomial factor is NP-hard.

1.3 Related Work

As indicated by the available theoretical results on computational hardness, polynomial time algorithms for the FFMSP can yield only solutions with no constant guarantee of approximation. Moreover, with growing instance size, complete techniques quickly reach their limits, and (meta-)heuristics become the only feasible approach for deriving high-quality solutions. The first attempt in the design of efficient metaheuristic approaches has been done in 2005 by Meneses et al. [12], who proposed a heuristic algorithm consisting of a simple greedy construction followed by an iterative improvement phase. Later, in 2007 Festa [5] designed a GRASP and more recently in 2012 Festa and Pardalos [6] proposed a genetic algorithm. In 2012, Mousavi et al. [13] devised a new function to be used as an alternative to the objective function when evaluating neighbor solutions during the local search phase of the algorithm proposed in [5]. In 2013, Ferone et al. [4] developed several pure and hybrid multistart iterative heuristics, including

- ◇ a pure GRASP approach, inspired by [5];
- ◇ a GRASP making use of path-relinking for intensification;
- ◇ a pure variable neighborhood search (VNS) approach;
- ◇ a VNS using path-relinking for intensification;
- ◇ a GRASP that uses VNS to implement the local search phase;
- ◇ and a GRASP approach that uses VNS to implement the local search phase and path-relinking for intensification.

Among all the presented algorithms, the GRASP approach making use of path-relinking for intensification can currently regarded to be state of the art for obtaining very good solutions in a resonable amount of computation time. GRASP [3, 7–9] is a multistart heuristic, where at each iteration a solution is probabilistically generated biased by a greedy function. Subsequently, a local search phase is applied starting from the just built solution until a locally optimal solution is found. Repeated applications of the construction procedure yield diverse starting solutions for the local search and the best overall local optimal solution is returned as the result.

1.4 Organization of the Paper

The remainder of this article is organized as follows. In Section 2, we present the first linear integer programming formulation for the FFMSP. In Section 3, we describe a hybrid ant colony approach based on a combination of ant colony optimization and a mathematical programming solver. Computational results for both the linear integer programming model and the ACO approach are reported in Section 4. Finally, concluding remarks and an outlook to future work are given in Section 5.

2 A Linear Integer Programming Model

First, we developed a linear integer programming model for solving the FFMSP. This model works on two sets of binary variables. The first set contains for each position i ($i = 1, \ldots, m$) of a possible solution and for each character $c_j \in \Sigma$ ($j = 1, \ldots, k$) a binary variable $x_{i,c_j} \in \{0, 1\}$. The second set consists of a binary variable $y_r \in \{0, 1\}$ ($r = 1, \ldots, n$) for each of the n input sequences provided in set Ω. The linear integer program itself can then be stated as follows.

$$\textbf{max} \sum_{r=1}^{n} y_r \tag{5}$$

subject to:

$$\sum_{c_j \in \Sigma} x_{i,c_j} = 1 \quad \text{for } i = 1, \ldots, m \tag{6}$$

$$\sum_{i=1}^{m} x_{i,s_i^r} \leq m - t y_r \quad \text{for } r = 1, \ldots, n \tag{7}$$

$$x_{i,c_j}, y_r \in \{0, 1\}$$

Hereby, constraints (6) ensure that for each position i of a possible solution exactly one character from Σ is chosen. Moreover, constraints (7) ensure that y_r can only be set to 1 if and only if the number of differences between $s^r \in \Omega$ and the possible solution (as defined by the setting of the variables x_{i,c_j}) is at least t. Remember, in this context, that s_i^r denotes the character at position i in $s^r \in \Omega$.

3 The Proposed Approach

This section is dedicated to a description of the hybrid ACO approach that we developed for tackling the FFMSP. In the following we first deal with preliminaries, before the framework of the proposed algorithm is outlined.

3.1 Preliminaries

Just like any other learning-based algorithm, ACO generally faces difficulties in the case of an objective function characterized by large plateaus. Unfortunately, this is exactly the case of the FFMSP, because the range of possible objective function values—that is, $\{0, \ldots, n\}$—is rather small. Therefore, we decided to introduce a refined way for comparing between valid solutions as follows. First, given a valid solution s, set $Q^s \subseteq \Omega$ is indirectly defined as follows:

- $|Q^s| = \max\{|P^s| + 1, n\}$;
- $d_H(s', s) \geq d_H(s'', s)$ for all $s' \in Q^s$ and $s'' \in \Omega \setminus Q^s$.

In words, set Q^s consists of all sequences $s' \in \Omega$ that have at least Hamming distance t with respect to s. Moreover, in case $|P^s| < n$, set Q^s also contains the sequence $\hat{s} \in \Omega \setminus P^s$ that has, among all sequence in $\Omega \setminus P^s$, the largest Hamming distance with respect to s.

Given two valid solutions s and s', a comparison operator (denoted by $>_{lex}$) is used for comparing them. This operator is defined in the following way. It holds that $s >_{\text{lex}} s'$ if and only if

$$f(s) > f(s') \textbf{ or } \left(f(s) = f(s') \textbf{ and } \sum_{s'' \in Q^s} d_H(s'', s) > \sum_{s'' \in Q^{s'}} d_H(s'', s') \right) . \quad (8)$$

In other words, the comparison between s and s' is done in a lexicographical way with the original objective function value as a first criterion, and the sum of the Hamming distances of the solutions from the respective sets Q^s and $Q^{s'}$ with respect to s and s' as a second criterion. It is intuitively assumed that the solution for which this sum is greater is somehow closer to a solution with a higher objective function value.

The second aspect of this section concerns the definition of the pheromone model and the greedy information, both being of crucial importance for any ACO algorithm. The pheromone model $\mathcal{T}$ consists of a pheromone value τ_{i,c_j} for each combination of a position i ($i = 1, \ldots, m$) in a solution sequence, and a letter $c_j \in \Sigma$. Moreover, to each of these combinations is assigned a greedy value η_{i,c_j} which measures the desirability to assign letter c_j to position i. Intuitively, value η_{i,c_j} should be reverse-proportional to the number of occurences of letter c_j at position i in the n input sequences. That is,

$$\eta_{i,c_j} := \frac{(n - |\{s \in \Omega \mid s_i = c_j\}|)}{n} . \quad (9)$$

3.2 Algorithmic Framework

The general framework of the proposed algorithm (see Algorithm 1) is as follows. First of all, calls to function $\mathsf{RunACO}(s^{bs})$ are executed until either the time limit $tlim_{\text{ACO}}$ is reached, or the number of unsuccessful calls to $\mathsf{RunACO}(s^{bs})$,

Algorithm 1. (Hybrid) ACO for the FFMSP

1. **input:** Ω, Σ, t, $tlim_{\text{ACO}}$, $runlim_{\text{ACO}}$, $tlim_{\text{CPLEX}}$
2. s^{bs} := NULL
3. $runs_{\text{ACO}} := 0$
4. **while** time limit $tlim_{\text{ACO}}$ not reached **and** $runs_{\text{ACO}} < runlim_{\text{ACO}}$ **do**
5. $s :=$ RunACO(s^{bs})
6. **if** $s >_{\text{lex}} s^{bs}$ **then**
7. $s^{bs} := s$
8. $runs_{\text{ACO}} := 0$
9. **else**
10. $runs_{\text{ACO}} := runs_{\text{ACO}} + 1$
11. **end if**
12. **end while**
13. **if** time limit $tlim_{\text{CPLEX}}$ not reached **then**
14. $s^{bs} :=$ SolutionPolishing($s^{bs}, tlim_{\text{CPLEX}}$)
15. **end if**
16. **output:** the best-so-far solution s^{bs}

as counted by $runs_{\text{ACO}}$, has reached the limit $runlim_{\text{ACO}}$. Hereby, a call to RunACO(s^{bs}) is regarded as *unsuccessful* if the best-so-far solution s^{bs} was not improved during the run. Function RunACO(s^{bs}) takes the best solution found so far (s^{bs}) as input and applies an ACO algorithm (as outlined below) until the convergence of the pheromone values is reached.

After this first phase (lines 4–12 of Algorithm 1) the algorithm possibly applies a second phase in which the hybridization with a mathematical programming solver takes place. This hybrid algorithm component works as follows. The best solution obtained by the ACO-phase is given to the MIP-solver CPLEX as a starting solution (see, for example, page 531 of the IBM ILOG CPLEX V12.1 user manual: *Starting from a Solution: MIP Starts*). CPLEX is then used to apply *solution polishing* to the given starting solution until the computation time limit $tlim_{\text{CPLEX}}$ is reached (see, for example, page 521 of the IBM ILOG CPLEX V12.1 user manual: *Solution polishing*). This is done in function SolutionPolishing($s^{bs}, tlim_{\text{CPLEX}}$). Solution polishing can be seen as a black box local search based on branch & cut, with the aim to improve a solution, rather than proving optimality. The best solution found after this phase is provided as output of the algorithm.

Note that a hybrid algorithm is obtained by a setting of $tlim_{\text{ACO}} < tlim_{\text{CPLEX}}$. On the other side, the pure ACO approach can be applied by setting, for example, $tlim_{\text{CPLEX}} = 0$ and by assigning a very large value to $runlim_{\text{ACO}}$ (in order to avoid that the while-loop of line 4 is stopped before the computation time limit $tlim_{\text{ACO}}$ is reached).

3.3 The ACO Phase

In the following, we focus on the description of function RunACO(s^{bs}) of Algorithm 1, which is pseudo-coded in Algorithm 2. Note that the ACO-implementation

Algorithm 2. Function RunACO(s^{bs}) of Algorithm 1

```
1.  input: s^bs
2.  s^rb := NULL, cf := 0
3.  Initialize all pheromone values of T to 0.5
4.  while cf < 0.99 do
5.     s^ib := NULL
6.     for all n_a artificial ants do
7.        s := ConstructSolution(T)
8.        s := LocalSearch(s)
9.        if s >_lex s^ib then s^ib := s
10.    end for
11.    s^ib := PathRelinking(s^ib, s^bs)
12.    if s^ib >_lex s^rb then s^rb := s^ib
13.    if s^ib >_lex s^bs then s^bs := s^ib
14.    ApplyPheromoneUpdate(cf,T,s^ib,s^rb)
15.    cf := ComputeConvergenceFactor(T)
16. end while
17. output: the best-so-far solution s^bs
```

used in this function is very similar to a $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (MMAS) implemented in the Hyper-Cube Framework (HCF) [14, 1]. The function takes as input the best-so-far solution s^{bs}. At the start, first, all pheromone values of $\mathcal{T}$ are initialized to 0.5. Then, at each iteration, $n_a = 10$ solutions are constructed in function ConstructSolution($\mathcal{T}$) on the basis of pheromone and greedy information. Local search is applied to each of these solutions in function LocalSearch(). Finally, path-relinking is applied to the best solution s^{ib} (after local search) generated in the current iteration. Solution s^{ib} is also referred to as the *iteration-best* solution. Path-relinking is applied in function PathRelinking(s^{ib}, s^{bs}). Hereby, s^{ib} serves as initial solution and the best-so-far solution s^{bs} serves as guiding solution. After updating (if necessary) the best-so-far solution and the so-called *restart-best* solution, which is the best solution found so far during the current application of RunACO(), the pheromone update is performed in function ApplyPheromoneUpdate(cf,$\mathcal{T}$,s^{ib},s^{rb}). Finally, at the end of each iteration, the new convergence factor value cf is computed in function ComputeConvergenceFactor($\mathcal{T}$). In the following we outline the working of the different functions in more detail.

ConstructSolution($\mathcal{T}$): For the construction of a new solution s, a letter from Σ is chosen successively for all positions from 1 to m. The letter choice for each position $1 \leq i \leq m$ is hereby performed as follows. First, for each letter $c_j \in \Sigma$ a choice probability $\mathbf{p}_{i,c_j}$ for position i is calculated as follows:

$$\mathbf{p}_{i,c_j} := \frac{\tau_{i,c_j} \cdot \eta_{i,c_j}}{\sum_{c \in \Sigma} \tau_{i,c} \cdot \eta_{i,c}} \tag{10}$$

Then, a value z is chosen uniformly at random from $[0.5, 1.0]$. In case $z \leq d_{rate}$, the letter $c \in \Sigma$ with the largest probability value is chosen for position i. Otherwise, a letter $c \in \Sigma$ is chosen randomly with respect to the probability values. Hereby, d_{rate} is a parameter of the algorithm, which was set to 0.8 after tuning by hand.

LocalSearch(s) and PathRelinking(s^{ib}, s^{bs}): The local search and path-relinking procedures used in this work are the same as the ones developed in the context of the GRASP algorithm published in [4]. Therefore, we only provide a short text-based description. For the pseudo-code we refer the interested reader to the original work.

The neighborhood of a solution s considered for the local search consists in all solutions that can be obtained by ex-changing exactly one character with a different one. The positions of a solution s are examined in a certain order, and, as soon as an improving neighbor is found, this neighbor is accepted as new current solution. The procedure stops once no improving neighbor can be found. The order in which positions are examined is randomly chosen for each application of local search, which is actually the only difference of our implementation in comparison to the one as given in [4].

Path-relinking is a concept initially proposed in [10] for the search intensification between two solutions. The path-relinking procedure that we applied in function PathRelinking(s^{ib}, s^{bs}) works roughly as follows. Solution s^{ib} is used as initial solution, and the best-so-far solution s^{bs} as guiding solution. The positions that both solutions have in common remain untouched. The solution space spanned by the positions that are different in the two solutions is explored by generating a path in the solution space linking solution s^{ib} with s^{bs}. The best solution found on this path is provided as output of path-relinking.

ApplyPheromoneUpdate(cf,$\mathcal{T}$,s^{ib},s^{rb}): The ACO procedure makes use of at most two different solutions for updating the pheromone values, namely solutions s^{ib} and s^{rb}. The weight of each solution for the pheromone update is determined as a function of cf, the convergence factor. The pheromone values τ_{i,c_j} are updated as follows:

$$\tau_{i,c_j} := \tau_{i,c_j} + \rho \cdot (\xi_{i,c_j} - \tau_{i,c_j}) \ , \tag{11}$$

where

$$\xi_{i,c_j} := \kappa_{ib} \cdot \Delta(s_i^{ib}, c_j) + \kappa_{rb} \cdot \Delta(s_i^{rb}, c_j) \ . \tag{12}$$

Hereby, function $\Delta(s, c_j)$ evaluates to 1 in case character c_j is to be found at position i of solution s. Otherwise, the function evaluates to 0. Moreover, κ_{ib} is the weight of solution s^{ib} and κ_{rb} the one of s^{rb}. It is required that $\kappa_{ib} + \kappa_{rb} = 1$. The weight values that we chose are the standard ones shown in Table 1. Finally, note that the algorithm works with upper and lower bounds for the pheromone values, that is, $\tau_{\max} = 0.999$ and $\tau_{\min} = 0.001$. In case a pheromone values surpasses one of these limits, the value is set to the corresponding limit. This has the effect that a complete convergence of the algorithm is avoided.

Table 1. Setting of κ_{ib} and κ_{rb} depending on the convergence factor cf

	$cf < 0.4$	$cf \in [0.4, 0.6)$	$cf \in [0.6, 0.8)$	$cf \geq 0.8$
κ_{ib}	1	2/3	1/3	0
κ_{rb}	0	1/3	2/3	1

ComputeConvergenceFactor($\mathcal{T}$): The formula that was used for computing the value of the convergence factor is as follows:

$$cf := 2 \left(\left(\frac{\sum_{\tau \in \mathcal{T}} \max\{\tau_{\max} - \tau, \tau - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right)$$

This implies that at the start of function RunACO(s^{bs}), cf has value zero. On the other side, in the case in which all pheromone values are either at $\tau_{\min}$ or at $\tau_{\max}$, cf has a value of one. In general, cf moves in $[0, 1]$. This completes the description of the proposed algorithm.

4 Experimental Evaluation

We implemented the proposed algorithm in ANSI C++ using GCC 4.7.3 for compiling the software. Moreover, the mathematical program outlined in Section 2 was solved with IBM ILOG CPLEX V12.1. The same version of CPLEX was used within the hybrid ACO approach. The experimental results that we outline in the following were obtained on a cluster of PCs with "Intel(R) Xeon(R) CPU 5160" CPUs of 4 nuclii of 3000 MHz and 4 Gigabyte of RAM.

4.1 Problem Instances

For the purpose of comparing to the state of the art, the algorithms proposed in this paper were applied to the set of benchmark instances introduced in [4]. This set consists of random instances of different size. More specifically, the number of input sequences (n) is in $\{100, 200\}$, and the length of the input sequences (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. Finally, as in [4], our algorithms were applied to all instances for different settings of parameter t. In particular, $t \in \{0.75m, 0.8m, 0.85m\}$.

4.2 Results

We applied a pure version of ACO ($tlim_{\text{ACO}} = 90s$, $tlim_{\text{CPLEX}} = 0s$, $runlim_{\text{ACO}} =$ very large integer) and a hybrid version of ACO ($tlim_{\text{ACO}} = 60s$, $tlim_{\text{CPLEX}} = 90s$, $runlim_{\text{ACO}} = 10$) exactly once to each problem instances. This was done for each of the three considered values for t. The (total) computation time limit

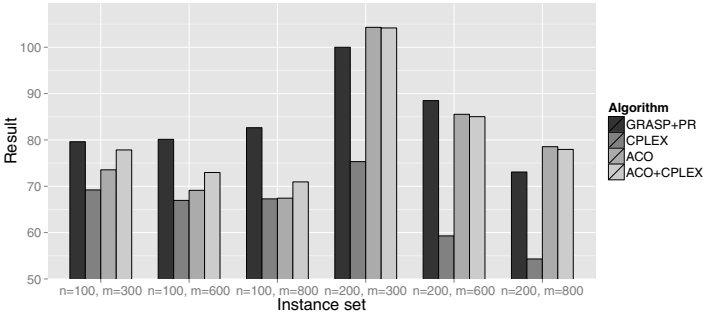
Table 2. Numerical results

n, m, t	GRASP+PR	CPLEX	ACO	HyACO
100, 300, $t = 0.75m$	**100**	**100**	**100**	**100**
100, 300, $t = 0.8m$	**79.61**	69.21	73.55	77.84
100, 300, $t = 0.85m$	13.18	22.08	24.64	**28.3**
100, 600, $t = 0.75m$	**100**	**100**	**100**	**100**
100, 600, $t = 0.8m$	**80.13**	66.95	69.12	72.97
100, 600, $t = 0.85m$	4.98	19.38	20.82	**22.82**
100, 800, $t = 0.75m$	**100**	**100**	**100**	**100**
100, 800, $t = 0.8m$	**82.64**	67.28	67.43	70.94
100, 800, $t = 0.85m$	1.84	18.3	19.84	**21.66**
200, 300, $t = 0.75m$	**200**	**200**	199.38	**200**
200, 300, $t = 0.8m$	100	75.32	**104.3**	104.17
200, 300, $t = 0.85m$	11.9	19.16	27.1	**28.59**
200, 600, $t = 0.75m$	**200**	**200**	199.99	**200**
200, 600, $t = 0.8m$	**88.49**	59.29	85.53	85.02
200, 600, $t = 0.85m$	2.42	18.12	21.03	**21.9**
200, 800, $t = 0.75m$	**200**	**200**	199.99	**200**
200, 800, $t = 0.8m$	73.08	54.31	**78.54**	77.95
200, 800, $t = 0.85m$	0.21	18.56	19.14	**20.4**

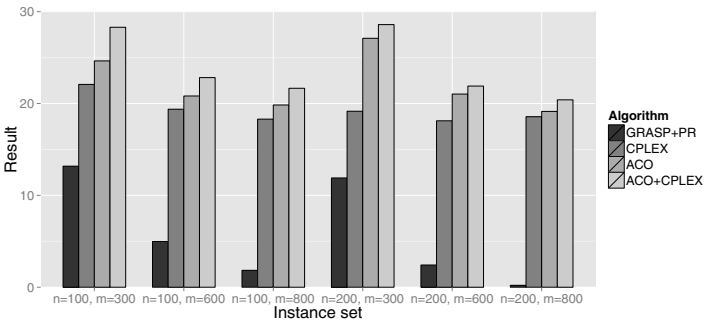
for each run was set to 90 CPU seconds in order to be comparable to the results reported for the GRASP hybridized with path-relinking in [4]. These three algorithms are henceforth referred to by ACO, HyACO, and GRASP+PR. Moreover, the mathematical programming solver CPLEX was applied with a CPU time limit of 90 seconds to each problem instance. Note that we report the values of the best feasible solutions found by CPLEX within the available computation time. The corresponding solutions are (expect for the case $t = 0.75m$) not proven to be optimal ones. The results are reported in Table 2. Note that for each combination of n, m, and t the given values are averages over the results for 100 problem instances. In each case, the best results are shown in bold font. For the analysis of the results, the three cases resulting from $t \in \{0.75m, 0.8m, 0.85m\}$ are treated seperately. This is because the behaviour of the algorithms changes greatly from one case to another.

Case $t = 0.75m$: This case results clearly in being the easiest one for all algorithms. In fact, GRASP+PR, CPLEX and HyACO find optimal solution to all 600 problem instances. The only algorithm that has slight difficulties when $n = 200$ is the pure ACO approach, which does not find an optimal solution in a few cases.

Case $t = 0.8m$: In this case, CPLEX is not able to prove optimality for any of the 600 instances. In fact, especially for the larger instances—that is, instances with $n = 200$—the performance of CPLEX decreases strongly. Concerning a comparison of the two ACO version the following can be observed. HyACO appears to outperform ACO consistently for the smaller instances ($n = 100$). This is not the case anymore for the larger instances, where the performance of the



(a) Results for all instances with $t = 0.8m$.



(b) Results for all instances with $t = 0.85m$.

Fig. 1. Graphical representation of the results from Table 2 for $t = 0.8m$ (see (a)) and $t = 0.85m$ (see (b))

two algorithms is comparable. GRASP+PR outperforms both ACO approaches on the set of smaller instances. However, this seems to change with growing instance size. In fact, when $n = 200$ and $m \in \{300, 800\}$ both ACO approaches outperform GRASP+PR. The results for this case are also shown in graphical form in Figure 1(a).

Case $t = 0.85m$: The results for this case are quite surprising. In fact, comparing the results of GRASP+PR with the results of the three new approaches, it turns out that GRASP+PR is not at all able to solve this case (see also the graphical representation of the results in Figure 1(b). Concerning a comparison of the three new approaches, it appears that HyACO consistently outperforms the pure ACO approach, which, in turn, consistently outperforms CPLEX.

5 Conclusions and Future Work

In this work we introduced a pure and a hybrid ACO approach for tackling the so-called *far from most string* problem. Moreover, we developed the first linear integer programming model for this problem. Finally, we described an experimental evaluation and a comparison to the state of the art from the literature,

which is a GRASP algorithm extended with path-relinking. The comparison showed that especially the hybrid ACO approach is very competitive, outperforming GRASP in particular for one of the three considered problem cases.

Future lines of work will be focused in particular on the one problem case (out out three) in which the ACO approaches seem to have slight disadvantages with respect to GRASP. In addition, we plan to test out approach on a larger set of instances possibly including real-world instances.

Acknowledgments. This work was supported by grant TIN2012-37930 of the Spanish Government. In addition, support is acknowledged from IKERBASQUE, the Basque Foundation for Science.

References

1. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE Transactions on Man, Systems and Cybernetics – Part B 34(2), 1161–1172 (2004)
2. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
3. Feo, T., Resende, M.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. 8, 67–71 (1989)
4. Ferone, D., Festa, P., Resende, M.: Hybrid metaheuristics for the far from most string problem. In: Blesa, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (eds.) HM 2013. LNCS, vol. 7919, pp. 174–188. Springer, Heidelberg (2013)
5. Festa, P.: On some optimization problems in mulecolar biology. Mathematical Bioscience 207(2), 219–234 (2007)
6. Festa, P., Pardalos, P.: Efficient solutions for the far from most string problem. Annals of Operations Research 196(1), 663–682 (2012)
7. Festa, P., Resende, M.: GRASP: An annotated bibliography. In: Ribeiro, C., Hansen, P. (eds.) Essays and Surveys on Metaheuristics, pp. 325–367. Kluwer Academic Publishers (2002)
8. Festa, P., Resende, M.: An annotated bibliography of GRASP – Part I: Algorithms. International Transactions in Operational Research 16(1), 1–24 (2009)
9. Festa, P., Resende, M.: An annotated bibliography of GRASP – Part II: Applications. International Transactions in Operational Research 16(2), 131–172 (2009)
10. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics 39, 653–684 (2000)
11. Lanctot, J., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Information and Computation 185(1), 41–55 (2003)
12. Meneses, C., Oliveira, C., Pardalos, P.: Optimization techniques for string selection and comparison problems in genomics. IEEE Engineering in Medicine and Biology Magazine 24(3), 81–87 (2005)
13. Mousavi, S., Babaie, M., Montazerian, M.: An improved heuristic for the far from most strings problem. Journal of Heuristics 18, 239–262 (2012)
14. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)