A Comprehensive Comparison of Metaheuristics for the Repetition-Free Longest Common Subsequence Problem.

Christian Blum[1] Maria J. Blesa[2]

[1] Artificial Intelligence Research Institute (IIIA-CSIC)
UAB Campus, Bellaterra, Spain
(christian.blum@iiia.csic.es)

[2] Computer Science Department
Technical University of Barcelona – BarcelonaTech
UPC North Campus, Barcelona, Spain
(mjblesa@cs.upc.edu)

Abstract

This paper deals with a well-known NP-hard string problem from the bio-informatics field: the repetition-free longest common subsequence problem. This problem has enjoyed an increasing interest in recent years, which has resulted in the application of several pure as well as hybrid metaheuristics. However, the literature lacks a comprehensive comparison between those approaches. Moreover, it has been shown that general purpose integer linear programming solvers are very efficient for solving many of the problem instances that were used so far in the literature. Therefore, in this work we extend the available benchmark set, adding larger instances to which integer linear programming solvers cannot be applied anymore. Moreover, we provide a comprehensive comparison of the approaches found in the literature. Based on the results we propose a hybrid between two of the best methods which turns out to inherit the complementary strengths of both methods.

Keywords: Repetition-free Longest Common Subsequence, Hybrid Metaheuristics, Matheuristic

1 Introduction

Longest common subsequence problems are string problems that arise frequently in bio-informatics applications. The most general problem from this class of problems is simply known as the *longest common subsequence* (LCS) problem. A problem instance (S, Σ) consists of a set $S = \{s_1, s_2, \ldots, s_n\}$ of $n \geq 2$ input strings over a finite alphabet Σ. The goal is to find the longest possible subsequence of all strings in S. Hereby, a string t is called a subsequence of a string s, if t can be produced from s by possibly deleting characters. For example, agt can be produced from $addagta$ by deleting the two occurrences of letter d and the last two occurrences of letter a. Apart from applications in computational biology [13, 18, 14], the LCS problem finds applications, for example, in data compression and file comparison [20, 3]. The LCS problem was shown to be NP-hard [16] for general n. If n is a constant, the problem is polynomially solvable by dynamic programming [13]. Standard dynamic programming approaches for this problem require $O(l^n)$ of time and space, where l is the length of the longest input string and n is the number of strings. However, dynamic programming becomes quickly impractical when n grows.

The specific problem considered in this work is a restricted version of the LCS problem: the repetition-free longest common subsequence (RFLCS) problem. Given exactly two input strings x and y over a finite alphabet Σ, the goal is to find a longest common subsequence with the additional restriction that no letter of the alphabet may appear more than once. The problem was introduced in [2] as a comparison measure for two sequences of biological origin. Moreover, the problem was shown to be APX-hard in [2], which implies NP-hardness.

1.1 Related Work

A lot of research effort has been dedicated to the more general LCS problem. Notable works include the large neighborhood search approach from [10], a beam search approach [6], fast heuristics [23], dynamic programming-based approaches such as [22], a decomposition and extension approach aimed for problem instances with many input strings [17], and a hyper-heuristic approach [21], among others. In comparison much less research effort has been dedicated to solving the RFLCS problem. First heuristics were proposed in [1, 2]. In addition, in recent years three metaheuristic approaches were described in the literature for solving the RFLCS problem. The first one was the adaptation of the Beam-ACO algorithm [6] from the LCS problem to the RFLCS problem; see [5]. Beam-ACO is a hybrid metaheuristic than makes use of a tree search method known as beam search in the context of an ant colony optimization (ACO) algorithm. Furthermore, an evolutionary algorithm (EA) for the RFLCS problem can be found in [9]. Finally, a Construct, Merge, Solve & Adapt (CMSA) approach, which is a hybrid metaheuristic combining heuristic elements with the application of a general purpose integer linear programming (ILP) solver, was presented in [4]. This last approach generates, at each iteration, sub-instances to an original problem instance and solves these by means of an ILP solver. As outlined in more detail below in Section 1.2, due to a rudimentary (or even the lack of a) comparison between these methods, so far it was not clear what were the advantages, respectively the weaknesses, of these approaches.

Bonizzoni et al. [8] studied some variants of the RFLCS, such as the one in which some symbols are required to appear in the sought solution possibly more than once. They showed that these variants are also APX-hard and that, in some cases, the problem of deciding the feasibility of an instance is NP-complete.

1.2 Our Contribution

So far it was not clear which one of the proposed techniques is really the state-of-the-art method. There are several reasons for that. First, the Beam-ACO approach [6] was only compared to the heuristics from [1, 2]. The same is the case for the evolutionary algorithm from [9], which was published at about the same time as the Beam-ACO approach. While the Beam-ACO approach clearly outperformed the heuristics in all cases, this was not the case for the evolutionary algorithm. Finally, in [4] the proposed CMSA approach was compared to Beam-ACO and to the application of the ILP solver CPLEX to all available problem instances. In this context it was shown that many of the problem instances considered to date can easily be solved by CPLEX (in the context of the used ILP model). For the remaining ones CMSA outperformed Beam-ACO in many cases. Summarizing, the performance relation between the proposed methods is not yet clear and can only be determined by means of a comprehensive comparison on a set of benchmark instances containing larger instances which can not easily be solved by CPLEX. This is exactly what we provide in this paper.

In addition, we take profit from the lessons learnt by this comparison and devise a hybrid method combining the CMSA algorithm with elements of beam search in order to obtain an algorithm with the respective strengths of both Beam-ACO and CMSA. The experimental results show that this hybrid can currently be called the state-of-the-art algorithm for the RFLCS problem.

1.3 Organization

The remainder of this work is organized as follows. In Section 2, the standard ILP model for the RFLCS problem is provided. Furthermore, the three metaheuristics from the literature tackling the RFLCS problem are described in Section 3. The extended set of benchmark instances, the tuning of the algorithm parameters, and a comprehensive experimental evaluation is described in Section 4. Finally, conclusions and an outlook to future work are provided in Section 5.

2 ILP Model for the RFLCS Problem

The RFLCS problem can be expressed as an integer linear program (ILP) as described below. The following notations are required. The length of x is henceforth denoted by l_x and the length of y by l_y. Furthermore, we assume that the positions in x, respectively y, are numbered from 1 to l_x, respectively from 1 to l_y. The letter at position i of x, respectively position j of y, is denoted by $x[i]$, respectively $y[j]$. The set Z of binary variables that is required for the ILP model contains a binary variable $z_{i,j}$ for each combination of $i = 1, \ldots, l_x$ and $j = 1, \ldots, l_y$ such that $x[i] = y[j]$. Moreover, two variables $z_{i,j}$ and $z_{k,l}$ are said to be *in conflict*, if and only if either ($i < k$ and $j > l$) or ($i > k$ and $j < l$). Finally, for each letter $a \in \Sigma$, let $Z_a \subset Z$ be the subset of Z that contains all variables $z_{i,j}$ such that $x[i] = y[j] = a$. The RFLCS problem can then be expressed as the problem of selecting a maximal number of non-conflicting variables from Z provided that, among all variables representing a letter $a \in \Sigma$, at most one variable is chosen. Given these notations, the ILP is as follows.

$$\mathbf{max} \sum_{z_{i,j} \in Z} z_{i,j} \tag{1}$$

subject to:

$$\sum_{z_{i,j} \in Z_a} z_{i,j} \leq 1 \quad \text{for } a \in \Sigma \tag{2}$$

$$z_{i,j} + z_{k,l} \leq 1 \quad \text{for all } z_{i,j} \text{ and } z_{k,l} \text{ being in conflict} \tag{3}$$

$$z_{i,j} \in \{0, 1\} \quad \text{for } z_{i,j} \in Z \tag{4}$$

Hereby, constraints (2) ensure that each letter from the alphabet is chosen at most once, and constraints (3) ensure that selected variables are not in conflict.

3 Metaheuristics

In the following we describe the three existing (hybrid) metaheuristic approaches: CMSA from [4], Beam-ACO from [6], and the EA from [9].

3.1 Description of CMSA

The pseudo-code for the application of CMSA to the RFLCS problem is provided in Algorithm 1. In the context of this algorithm, both solutions to the problem and sub-instances are expressed as subsets of the complete set of variables (Z). The meaning of a variable $z_{i,j}$ being part of a solution S is that $z_{i,j}$ must be given value one in order to produce the corresponding solution. The main loop of CMSA, which is executed while the CPU time limit is not reached, consists of the following actions. First, both the best-so-far solution S_{bsf} and the restricted problem instance Z_{sub} are initialized to the empty set. Then, a number of n_a solutions is probabilistically constructed in function ProbabilisticSolution-Construction(Z); see line 6 of Algorithm 1. The variables that form part of these solutions are added to Z_{sub}. The age of a newly added variable $z_{i,j}$, as denoted by age[$z_{i,j}$], is initialized to zero. After the construction of n_a solutions, an ILP solver is applied with a computation time limit of $t_{\max}$ seconds to sub-instance

Algorithm 1 CMSA for the RFLCS problem

```
1: **input:** input strings $x$ and $y$, values for parameters $n_a$, $\mathrm{age}_{\max}$, $d_{\mathrm{rate}}$, $l_{\mathrm{size}}$
   and $t_{\max}$
2: $S_{\mathrm{bsf}} := \emptyset$, $Z_{\mathrm{sub}} := \emptyset$
3: $\mathrm{age}[z_{i,j}] := 0$ for all $z_{i,j} \in Z$
4: **while** CPU time limit not reached **do**
5:    **for** $i = 1, \ldots, n_a$ **do**
6:       $S :=$ ProbabilisticSolutionConstruction($Z$, $x$, $y$, $d_{\mathrm{rate}}$, $l_{\mathrm{size}}$)
7:       **for** all $z_{i,j} \in S$ **and** $z_{i,j} \notin Z_{\mathrm{sub}}$ **do**
8:          $\mathrm{age}[z_{i,j}] := 0$
9:          $Z_{\mathrm{sub}} := Z_{\mathrm{sub}} \cup \{z_{i,j}\}$
10:      **end for**
11:   **end for**
12:   $S'_{\mathrm{ilp}} :=$ ApplyILPSolver($Z_{\mathrm{sub}}$, $t_{\max}$)
13:   **if** $|S'_{\mathrm{ilp}}| > |S_{\mathrm{bsf}}|$ **then** $S_{\mathrm{bsf}} := S'_{\mathrm{ilp}}$
14:   Adapt($Z_{\mathrm{sub}}$, $S'_{\mathrm{ilp}}$, $\mathrm{age}_{\max}$)
15: **end while**
16: **output:** $S_{\mathrm{bsf}}$
```

Z_{sub} (line 12 of Algorithm 1). The best solution found by the ILP solver within the allowed computation time is denoted by S'_{ilp}. In case S'_{ilp} is better than the current best-so-far solution S_{bsf}, S'_{ilp} replaces S_{bsf} (line 13). Next, the current sub-instance Z_{sub} is subject to changes, based on solution S'_{ilp} and on the age values of the variables. This is done in function Adapt(Z_{sub}, S'_{ilp}, $\mathrm{age}_{\max}$) in line 14 as follows. First, the age of each variable in Z_{sub} is incremented, and, subsequently, the age of each variable in $S'_{\mathrm{ilp}} \subseteq Z_{\mathrm{sub}}$ is re-initialized to zero. Finally, those variables from Z_{sub} whose age has reached the age limit ($\mathrm{age}_{\max}$) are deleted from Z_{sub}. The motivation behind this mechanism is as follows. On the one side, variables which never appear in a solution generated by the ILP solver should be removed from Z_{sub} after a while, because they arguably slow down the ILP solver. On the other side, components which appear in optimal solutions seem to be useful and should therefore be maintained.

The last remaining component of the CMSA algorithm is the probabilistic construction of solutions in function ProbabilisticSolutionConstruction(Z, x, y, d_{rate}, l_{size}). Each solution construction starts with an empty solution $S = \emptyset$. The first step consists in generating the set of variables that serve as options to be added to S. More specifically, the initial set C is generated in order to contain for each letter $a \in \Sigma$ the variable $z_{i,j} \in Z_a$ (if any) such that $i < k$ and $j < l$, $\forall z_{k,l} \in Z_a$. Moreover, options $z_{i,j} \in C$ are given a weight value $w(z_{i,j}) := \frac{i}{l_x} + \frac{j}{l_y}$, which is a known greedy function for longest common subsequence problems. At each construction step, exactly one variable is chosen from C and added to S. For doing so, first, a value r is chosen uniformly at random from $[0, 1]$. In case $r \leq d_{\mathrm{rate}}$, where d_{rate} is a parameter of the algorithm, the variable

$z_{i,j} \in C$ with the smallest weight value is deterministically chosen. Otherwise, a candidate list $L \subseteq C$ of size $\min\{l_{\text{size}}, |C|\}$ containing the options with the lowest weight values is generated and exactly one variable $z_{i,j} \in L$ is then chosen uniformly at random and added to S. Note that l_{size} is another parameter of the solution construction process. Finally, the set of options C for the next construction step is generated. This is done such that C only contains variables that represent letters that are not already represented by one of the variables in S. Moreover, being $z_{i,j}$ the last variable that was added to S, C contains for each non-represented letter $a \in \Sigma$ the variable $z_{r,s} \in Z_a$ (if any) with the lowest weight value $w(z_{r,s})$ calculated as $w(z_{r,s}) := \frac{r-i}{l_x-i} + \frac{s-j}{l_y-j}$. The construction of a complete (valid) solution is finished when the set of options is empty.

3.2 Description of Beam-ACO

As mentioned before, Beam-ACO is a general algorithm that combines the algorithmic framework of ACO with a tree search method known as beam search. In the following we first describe the ACO-based framework of the Beam-ACO algorithm for the RFLCS problem. Afterwards, the beam search component is presented.

In contrast to CMSA, where solutions to the problem were kept in terms of the set of variables from the ILP model that have value one in the corresponding solution, solutions in Beam-ACO are represented in a different way. Solutions in this representation are henceforth called *ACO-solutions*. Any common subsequence t of strings x and y can be translated in a well-defined way into a unique ACO-solution $T = (X, Y)$, where both X and Y are binary strings and X is of length l_x while Y is of length l_y: first, the position of the left-most occurrence of $t[1]$ in x (where $t[1]$ is the first character of t) is determined, say k_1. Then, all $X[i]$ with $i < k_1$ are set to 0, while $X[k_1] := 1$. Next, the position of the first occurrence of $t[2]$ in x after position k_1 is determined, say k_2. Then, all $X[i]$ with $k_1 < i < k_2$ are set to 0, while $X[k_2] := 1$. This is continued until all positions of t are treated. Afterwards, the same procedure is applied to string y in order to produce Y.

Apart from the solution representation, another crucial component of any ACO algorithm is the so-called *pheromone model* $\mathcal{T}$, which is a probabilistic model used for generating solutions to the tackled problem. In the context of Beam-ACO for the RFLCS problem, $\mathcal{T}$ consists of a pheromone value $0 \leq \tau_{x,i} \leq 1$ for each position i of input sequence x ($1 \leq i \leq l_x$), and a pheromone value $\tau_{y,j}$ for each position j of input sequence y ($1 \leq j \leq l_y$). Observe that a pheromone value $\tau_{x,i}$ (respectively $\tau_{y,j}$) indicates the *desirability* of adding the letter at position i of string x (respectively, the letter at position j of string y) to the solution under construction.

Algorithm Framework. The algorithmic framework of the proposed Beam-ACO approach—see Algorithm 2 for the pseudo-code—is a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant

Algorithm 2 Beam-ACO for the RFLCS problem

```
1: input: input strings $x$ and $y$, values for parameters $k_{bw}$, $\mu \in \mathbb{Z}^+$, and $d_{\text{rate}}$
2: $T^{\text{bsf}} :=$ NULL, $T^{\text{rb}} :=$ NULL, $cf := 0$, $bs\_update :=$ FALSE
3: Initialize all pheromone values to 0.5
4: while CPU time limit not reached do
5:     $T^{\text{pbs}} :=$ ProbabilisticBeamSearch($k_{bw}$,$\mu$, $d_{\text{rate}}$) {see Alg. 3}
6:     if $|t^{\text{pbs}}| > |t^{\text{rb}}|$ then $T^{\text{rb}} := T^{\text{pbs}}$
7:     if $|t^{\text{pbs}}| > |t^{\text{bsf}}|$ then $T^{\text{bsf}} := T^{\text{pbs}}$
8:     ApplyPheromoneUpdate($cf$,$bs\_update$,$\mathcal{T}$,$T^{\text{pbs}}$,$T^{\text{rb}}$,$T^{\text{bsf}}$)
9:     $cf :=$ ComputeConvergenceFactor($\mathcal{T}$)
10:    if $cf > 0.99$ then
11:        if $bs\_update =$ TRUE then
12:            Re-init. all pheromone values to 0.5, $T^{\text{rb}} :=$ NULL, $bs\_update :=$ FALSE
13:        else
14:            $bs\_update :=$ TRUE
15:        end if
16:    end if
17: end while
18: output: the string version $t^{\text{bsf}}$ of $T^{\text{bsf}}$
```

System implemented in the hyper-cube framework (HCF) [7]. The following notations are required: (1) T^{bsf} is the *best-so-far* solution, that is, the best solution generated by the algorithm over time; (2) T^{rb} is the *restart-best* solution, that is, the best solution generated since the algorithm's last restart; (3) $0 \leq cf \leq 1$ is the *convergence factor*, which is a measure indicating the state of the convergence of the algorithm; and (4) *bs_update* is a Boolean control variable which assumes value *true* when the algorithm reaches convergence. The algorithm works as follows. Initially, the pheromone values are set to 0.5. Then, at each iteration, a probabilistic beam search based on pheromone values and greedy information is applied. For a description of the beam search component see Section 3.3. The result of the application of beam search is a solution T^{pbs}. Next, an update of the pheromone values is performed in ApplyPheromoneUpdate(cf, bs_update, $\mathcal{T}$, T^{pbs}, T^{rb}, T^{bsf}). Moreover, the value of the convergence factor cf is calculated. Depending on cf and the value of the Boolean variable *bs_update*, a decision on whether to restart the algorithm is made. In case of a restart, all pheromone values are set again to 0.5. As in the case of CMSA, the stopping criterion of the algorithm is a maximum CPU time. The algorithm provides the string version T^{bsf} of the best-so-far ACO-solution T^{bsf} as output. The two remaining procedures of Algorithm 2 are detailed in the following.

ApplyPheromoneUpdate(cf,bs_update,$\mathcal{T}$,T^{pbs},T^{rb},T^{bsf}): At each iteration, the three solutions T^{pbs}, T^{rb}, and T^{bsf} are used for updating the pheromone values. The impact/weight of each solution for this update is determined as a function of the convergence factor cf. The pheromone values $\tau_{x,i}$ corresponding to input

Table 1: Setting of κ_{pbs}, κ_{rb}, κ_{bs}, and ρ depending on the convergence factor *cf* and the Boolean control variable *bs_update*

	bs_update = FALSE				*bs_update*
	$cf < 0.4$	$cf \in [0.4, 0.6)$	$cf \in [0.6, 0.8)$	$cf \geq 0.8$	= TRUE
κ_{pbs}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1
ρ	0.2	0.2	0.2	0.15	0.15

string x are updated as follows:

$$\tau_{x,i} := \tau_{x,i} + \rho \cdot (\xi_{x,i} - \tau_{x,i}) \ , \tag{5}$$

where

$$\xi_{x,i} := \kappa_{pbs} \cdot X^{pbs}[i] + \kappa_{rb} \cdot X^{rb}[i] + \kappa_{bs} \cdot X^{bs}[i] \ . \tag{6}$$

Hereby, κ_{pbs} is the weight of solution $T^{\text{pbs}} = (X^{pbs}, Y^{pbs})$, κ_{rb} the one of $T^{\text{rb}} = (X^{rb}, Y^{rb})$, κ_{bs} the one of $T^{\text{bsf}} = (X^{bs}, Y^{bs})$, and $\kappa_{pbs}+\kappa_{rb}+\kappa_{bs} = 1$. The weight values chosen for the experimental evaluation are standard for ACO algorithms; see Table 1. Obviously, the pheromone update formulas above are also applied to the pheromone values $\tau_{y,j}$ corresponding to input string y. Finally, note that the algorithm works with upper and lower bounds for the pheromone values, that is, $\tau_{\max} = 0.999$ and $\tau_{\min} = 0.001$. Not letting the pheromone values pass these limits, has the effect that a complete convergence of the algorithm is avoided.

ComputeConvergenceFactor($\mathcal{T}$): The following is the formula used for calculating the value of the convergence factor.

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

Accordingly, when starting (or re-starting) the algorithm, *cf* has value zero, and when all pheromone values have either value $\tau_{\min}$ or $\tau_{\max}$, *cf* has value one. In general, *cf* moves in $[0, 1]$.

3.3 The Beam Search Component

The pseudo-code for the probabilistic beam search component is provided in Algorithm 3. Note that this pseudo-code shows the working of function ProbabilisticBeamSearch(k_{bw},μ, d_{rate}) of Algorithm 2. In the context of this algorithm, feasible solutions are represented as strings that are subsequences of the two input sequences. In general, solutions are constructed from left to right. Moreover, partial solutions are extended by appending exactly one letter at a time. The beam search component has two input parameters: (1) $k_{\text{bw}} \in \mathbb{Z}^+$,

Algorithm 3 Procedure ProbabilisticBeamSearch(k_{bw},μ, d_{rate}) of Algorithm 2

```
1:  input: input strings $x$ and $y$, values for parameters $k_{\text{bw}}$, $\mu \in \mathbb{Z}^+$, and $d_{\text{rate}}$
2:  $B_{\text{compl}} := \emptyset$, $B := \{\epsilon\}$, $t_{\text{bsf}} := \emptyset$
3:  while $B \neq \emptyset$ do
4:    $E_B :=$ Produce_Extensions($B$)
5:    $E_B :=$ Filter_Extensions($E_B$)
6:    $B := \emptyset$
7:    for $k = 1, \ldots, \min\{\lfloor \mu k_{\text{bw}} \rfloor, |E_B|\}$ do
8:      $za :=$ Choose_Extension($E_B, d_{\text{rate}}$)
9:      $t := za$
10:     if $\text{UB}(t) = |t|$ then
11:       $B_{\text{compl}} := B_{\text{compl}} \cup \{t\}$
12:       if $|t| > |t_{\text{bsf}}|$ then $t_{\text{bsf}} := t$ end if
13:     else
14:       if $\text{UB}(t) \geq |t_{\text{bsf}}|$ then $B := B \cup \{t\}$ end if
15:     end if
16:     $E_B := E_B \setminus \{t\}$
17:   end for
18:   $B :=$ Reduce($B, k_{\text{bw}}$)
19: end while
20: output: The ACO-version $T^{\text{pbs}}$ of argmax $\{|t| \mid t \in B_{\text{compl}}\}$
```

which is the so-called *beam width*, and (2) $\mu \in \mathbb{R}^+ \geq 1$, which is a parameter used to determine the maximal number of solution extensions that may be chosen at each step. The main data structure is the so-called *beam* B, which is a set for storing the current partial solutions. B is initialized with the empty string denoted by ϵ. Assuming that E_B denotes the set of all feasible extensions of the partial solutions in B, $\lfloor \mu k_{\text{bw}} \rfloor$ of these extensions are selected at each step based on a greedy function and the pheromone values. When choosing an extension from E_B, it is stored in B_{compl} in case it corresponds to a complete—that is, non-extensible—solution. However, if the chosen extension corresponds to a partial solution and its upper bound value (computed by function UB()) is greater than the length of the best-so-far solution t_{bsf}, it is stored in the new beam B of the next step of the algorithm. In order to finalize a step, B must be reduced in case it contains more than k_{bw} partial solutions. This is done on the basis of the upper bound values, that is, the best partial solutions with respect to the upper bound values remain in B. The four procedures of Algorithm 3 are outlined in detail in the following.

Produce_Extensions(B): This procedure generates the set E_B of non-dominated extensions of all the partial solutions in B. This is done as follows. First, given a partial solution t, the reduced alphabet Σ^t only contains letters which do not appear in t. Furthermore, let $x = x^+ \cdot x^-$ be the partition of input sequence x into substrings x^+ and x^- such that t is a subsequence of x^+, and x^- has maximal length. In the same way, y^+ and y^- are defined with respect to input string y. Given this partition, which is well-defined, *position pointers* $p_x := |x^+|$ and $p_y := |y^+|$ are introduced. Moreover, the position of the first appearance of a letter $a \in \Sigma^t$ in strings x and y after the position pointers p_x and p_y is well-defined and denoted by p_x^a and p_y^a. In case letter $a \in \Sigma^t$ does not appear in x (respectively y), p_x^a (respectively p_y^a) is set to ∞. In this context, a letter $a \in \Sigma^t$ is called *dominated*, if there exists at least one letter $b \in \Sigma^t$, $a \neq b$, such that $p_x^b < p_x^a$ and $p_y^b < p_y^a$. Finally, $\Sigma^t_{\text{nd}} \subseteq \Sigma^t$ denotes the set of non-dominated letters of the reduced alphabet Σ^t with respect to partial solution t. Observe also that letters in Σ_t^{nd} are required to appear at least once in both x^- and y^-. Finally, set E_B is generated as the set of subsequences ta, where $t \in B$ and $a \in \Sigma^t_{\text{nd}}$.

Filter_Extensions(E_B): The non-domination relation—as defined above—can also be considered for extensions of different partial solutions of the same length. Formally, given two extensions $ta, zb \in E_B$, where $t \neq z$ but not necessarily $a \neq b$, ta is said to dominate zb if and only if the position pointers concerning a appear before the position pointers concerning b in the corresponding remaining parts of the two input strings. Using this relation, E_B is filtered in order to remove all dominated elements.

Choose_Extension(E_B, d_{rate}): The probabilistic choice of a partial solution from E_B is made both on the basis of a greedy function and the pheromone values. The greedy value of an extension $ta \in E_B$ is computed as follows:

$$\eta(ta) := \left(\frac{p_x^a - p_x}{|x^-|} + \frac{p_y^a - p_y}{|y^-|} \right)^{-1} \tag{7}$$

Note that this is the same greedy function as the one used in the context of the CMSA algorithm for generating solutions in a probabilistic way. However, instead of directly using these greedy values, it was decided to use the corresponding ranks. More specifically, the final greedy value $\nu(ta)$ of a partial solution $ta \in E_B$ is calculated as the sum of the ranks of the greedy weights that correspond to the construction steps that were performed to construct string ta. With this definition of $\nu()$, the probability for each $ta \in E_B$ can be defined as follows:

$$\mathbf{p}(ta) = \frac{\left(\min\{\tau_{x,p_x^a}, \tau_{y,p_y^a}\} \cdot \nu(ta)^{-1}\right)}{\sum\limits_{zb \in E_B} \left(\min\{\tau_{x,p_x^b}, \tau_{y,p_y^b}\} \cdot \nu(zb)^{-1}\right)} \tag{8}$$

The intuition for this formula is as follows: If at least one of the pheromone values τ_{x,p_x^a} and τ_{y,p_y^a} is low, the corresponding letter should not yet be appended to the string, because there exists another letter that should be appended first. Finally, each application of function Choose_Extension(E_B) is either executed probabilistically, or deterministically (by choosing the option with the highest probability). The probability for a deterministic choice, also called the *determinism rate*, is henceforth denoted by $d_{\mathrm{rate}} \in [0,1]$.

$\mathsf{Reduce}(B, k_{\mathrm{b}w})$: In this procedure, the new beam B is reduced, if necessary, to exactly $k_{\mathrm{b}w}$ elements. This is done based on the upper bound values. Given a partial solution $t \in B$, $\delta(x,a)$ (for all $a \in \Sigma^t$) evaluates to one, in case letter a appears at least once in x^-. Otherwise, $\delta(x,a)$ evaluates to zero. The same holds for $\delta(y,a)$. The upper bound value of $t \in B$ is then defined as follows:

$$\mathrm{UB}(t) := |t| + \sum_{a \in \Sigma^t} \min\{\delta(x,a), \delta(y,a)\} \tag{9}$$

It is easy to see that the upper bound value of any partial solution can be computed in linear time.

3.4 Description of the Evolutionary Algorithm

In the following we describe our re-implementation of the evolutionary algorithm (EA) from [9]. Note that, due to ambiguities and possible errors in the original description—which could not even be resolved by conversation with the authors of [9]—we were forced to take a couple of design decisions concerning (1) the selection operator (tournament selection) and (2) the calculation of the so-called frequency vector for resolving conflicts (further details are given below). Nevertheless, we made sure that the results of our re-implementation are not worse than those presented in the original paper.

Before delving into the algorithm description, we will deal with some key issues of the EA, namely, the representation of solutions, the repair of invalid solutions, and the fitness function.

Solution representation. The EA works on individuals that are binary strings of the length of input string x, that is, l_x. The *candidate solution* z corresponding to an individual I is produced as follows. First, input string x is copied, that is, $z := x$. Then, all letters at positions where I has value zero are deleted from z. The remaining string is the candidate solution. Observe that a candidate solution obtained in this way does not necessarily guarantee that the "repetition-free" constraint is being respected. In such a case the corresponding individual I from which z was obtained is said to suffer from *conflicts*. More specifically, an individual I is said to suffer from a conflict if and only if $I[j] = I[k] = 1$ and $x[j] = x[k]$, for some $j \neq k$ such that $1 \leq j,k \leq l_x$. In other words, an individual is said to suffer from conflicts if and only if the corresponding candidate solution contains at least one repetition of at least one of the characters of the alphabet Σ.

As an example, let us consider input string $x = \mathrm{ACGAGT}$ and an individual $I = 101101$. Note that x and I have the same length. The corresponding candidate solution is $z = \mathrm{AGAT}$. As z has two appearences of letter A, the individual I is said to suffer from conflicts.

Repair of invalid individuals. When given an individual with conflicts, these conflicts are resolved in one of the following two ways. Conflicts of the individuals in the initial population are randomly resolved, which means that, among all the repetitions of the same character in an individual I, one is randomly chosen to be maintained and the rest of the positions that refer to the same character are given value zero. In the example from above, the repaired individual would be either 001101 or 101001. All other individuals—from populations other than the initial one—that suffer from conflicts are repaired by using a so-called frequency vector V of length l_x, whose positions take values between 0 and 1. Such a frequency vector V is obtained on the basis of the valid solutions of the current population as outlined in detail below. For any two conflicting positions j and k of an individual I, if $V[j] > V[k]$ then $I[k]$ is set to zero. In the opposite case, $I[j]$ is set to zero. Considering again the example from above, and assuming to have frequency vector $V = (0.6, 0.0, 0.1, 0.4, 0.9, 1.0)$, individual $I = 101101$ would be repaired by setting the fourth position to zero.

Fitness function. Let I be a (conflict-free) individual, and let z be the corresponding candidate solution produced as described above. The fitness $f(I)$ of such an individual I is defined as the length of the LCS between z and y (which is the second input string), penalized by a factor that considers the number of characters in z that do not form part of the LCS between z and y. In the following, let $LCS(z, y)$ denote the LCS between z and y, which can be computed efficiently by the dynamic programming algorithm of Smith and Waterman [19]. Moreover, let $|LCS(z, y)|$ denote the length of $LCS(z, y)$. Then,

$$f(I) := |LCS(z, y)| - \sum_{j=1}^{l_x} \phi(z[j], y) \tag{10}$$

where

$$\phi(z[j], y) := \begin{cases} 1 & \text{if } z[j] \notin LCS(z, y) \\ 0 & \text{if otherwise.} \end{cases} \tag{11}$$

Note that—as I must be a conflict-free individual—the LCS between z and y also corresponds to a repetition-free longest common subsequence of the input strings x and y. As an example, consider again a problem instance where x = ACGAGT. Moreover, the second input string (y) is as follows: y = AGTCC. Let us consider the conflict-free individual I = 001101, which corresponds to candidate solution z = GAT. The LCS between z and y is GT. Therefore, $|LCS(z, y)| = 2$. This means there is one position of z which does not contribute to the LCS between z and y. Therefore, $f(I) := 2 - 1 = 1$.

After describing these important algorithmic aspects, we provide a description of the pseudo-code of EA in Algorithm 4. As input the algorithm takes the two input strings, the population size p_{size}, the crossover rate c_{rate} and the muta-

Algorithm 4 (Modified) EA for the RFLCS problem from [9]

```
1: input: input strings $x$ and $y$, values for parameters $p_{\text{size}}$, $c_{\text{rate}}$, and $m_{\text{rate}}$
2: $P :=$ CreateInitialPopulation($p_{\text{size}}$)
3: while CPU time limit not reached do
4:    $V :=$ ComputeFrequencyVector($P$)
5:    $P' := \emptyset$
6:    while $|P'| < p_{\text{size}}$ do
7:       $\{I_1, I_2\} :=$ TournamentSelection($P$)
8:       $\{I'_1, I'_2\} :=$ Crossover($I_1$, $I_2$, $c_{\text{rate}}$)
9:       $\{I'_1, I'_2\} :=$ Mutate($I'_1$, $I'_2$, $m_{\text{rate}}$)
10:      $\{I'_1, I'_2\} :=$ SolveConflicts($I'_1$, $I'_2$, $V$)
11:      $P' := P' \cup \{I'_1, I'_2\}$
12:   end while
13:   $I_{\text{bsf}} := \text{argmax}\{f(I) \mid I \in P\}$
14:   $P' := P' \cup I_{\text{bsf}}$
15:   Remove the worst individuals from $P'$ until $|P'| = p_{\text{size}}$
16:   $P := P'$
17: end while
18: $I_{\text{bsf}} := \text{argmax}\{f(I) \mid I \in P\}$
19: output: The solution corresponding to $I_{\text{bsf}}$
```

tion rate m_{rate}.[1] The procedure CreateInitialPopulation(p_{size}) (line 2) generates an initial population of p_{size} individuals at random. Each of these individuals has an equal probability (i.e., 0.5) of having a zero or a one at each position. As stated before, the length of each individual is l_x. Moreover, conflicts in individuals are randomly resolved. At each iteration, the algorithm first applies method ComputeFrequencyVector(P) (line 4). Based on the current population P, this method computes the frequency vector V of length l_x which is used to resolve conflicts in individuals.[2] In particular, V is computed as follows. First, a set Q of $p_{\text{size}}/2$ individuals are selected from P by means of tournament selection using a tournament size of $|P|/2$. Then, each position $1 \leq i \leq l_x$ of V is set to $\frac{n_i}{|Q|}$, where n_i is the number of individuals in Q that have a one at position i.

After the computation of V, p_{size} new individuals are generated in lines 6–12 of Algorithm 4 by means of the application of the classical operators of evolutionary algorithms: selection, crossover and mutation. The selection operator selects two individuals I_1 and I_2 from the current population P. This is done in method TournamentSelection(P) (line 7) by means of tournament selection. The crossover operator (method Crossover(I_1, I_2, c_{rate}), line 8) implements a standard one-point crossover in which the crossover point is randomly chosen.

[1]Default parameter values used for the experiments in [9]: $p_{\text{size}} = 100$, $c_{\text{rate}} = 0.9$, $m_{\text{rate}} = 0.05$.

[2]According to the description as given in [9], this vector is re-computed inside the while-loop (lines 6–12). In our opinion, this is a description error, because re-computing V at every iteration would only introduce minor variations to V. Therefore, we decided to re-compute V only once per main iteration of the algorithm.

This is done with probability c_{rate}. Otherwise, the two individuals are simply copied. The mutation operator (method Mutate(I'_1, I'_2, m_{rate}), line 9) flips, with probability m_{rate}, each bit in individuals I'_1 and I'_2 resulting from the crossover. Before adding the created offspring to the new population P' under construction (line 11), the procedure SolveConflicts(I'_1, I'_2, V) (line 10) resolves the conflicts that the crossover and mutation operators might have caused. In contrast to resolving conflicts in the individuals of the initial population, here the conflicts are resolved using the frequency vector V as described above.

Once P' contains at least p_{size} individuals, the best individual of P is explicitly added to P' in order to guarantee the survival of the best individual from one generation to the next (elitism). Moreover, the worst individuals are removed from P' in order to keep P' of size p_{size} (see lines 13 and 14). Finally the current population P is replaced with the new population P' (line 16).

The whole process (lines 4–16) is iterated while the CPU time limit is not reached. The output of the algorithm is the best solution found.

4 Experimental Evaluation

The three algorithms presented in Section 3—henceforth denoted by Cmsa, Beam-Aco, and Ea—were implemented in ANSI C++ using GCC 4.7.3, without the use of any external libraries. In addition the ILP models concerning all problem instances were solved with the ILP solver IBM ILOG CPLEX v12.2 in one-threaded mode.[3] The same version of CPLEX was used for solving the reduced ILP models in the context of CMSA. Note that the performance of CPLEX is always measured in relation to the used ILP model. Therefore, when referring to the performance of CPLEX, we always refer to the performance of CPLEX in relation to the ILP model provided in Section 2. In order to indicate this, the application of CPLEX to this ILP model is henceforth denoted by Cplex*.

The experimental evaluation has been performed on a cluster of PCs with Intel(R) Xeon(R) CPU 5670 CPUs of 12 nuclei of 2933 MHz and at least 40 Gigabytes of RAM. The remainder of this section is organized as follows. First, the extended set of benchmark instances is described. Second, the tuning experiments that were conducted in order to determine a proper setting for the parameters of the three metaheuristics are outlined. Finally, an exhaustive experimental evaluation is presented.

4.1 Problem Instances

The algorithms that were so-far proposed in the literature were all evaluated on the following two sets of benchmark instances originally proposed in [2]. The

[3]IBM ILOG CPLEX is an optimization software package which includes state-of-the-art exact techniques for solving integer linear programming models to optimality. It is available for free for academic purposes. For more information we refer the interested reader to `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`.

first set—referred to as Set1—consists, for each combination of input sequence length $n \in \{32, 64, 128, 256, 512\}$ and alphabet size

$$|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\},$$

of exactly 10 randomly generated problem instances. The second set of instances—referred to as Set2—is generated on the basis of alphabet sizes $|\Sigma| \in \{4, 8, 16, 32, 64\}$ and the maximal repetition of each letter $rep \in \{3, 4, 5, 6, 7, 8\}$ in each input string. For each combination of $|\Sigma|$ and rep this instance set consists of 10 randomly generated problem instances. Provided that Cplex* was found to be very efficient for solving most of the instances from these two sets, the following extension of Set2 was considered in [4]. This extension contains for each combination of $|\Sigma| \in \{128, 256\}$ and $rep \in \{3, 4, 5, 6, 7, 8\}$ 10 randomly generated problem instances. In this work we, first, extend the above-mentioned sets from 10 to 30 problem instances for each parameter combination. Second, we extend Set2 with 30 problem instances for each combination of $|\Sigma| = 512$ and $rep \in \{3, 4, 5, 6, 7, 8\}$. Third, Set1 is also extended with 30 problem instances for each combination of input sequence length $n \in \{1024, 2048, 4096\}$ and alphabet size $|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\}$. This makes a total of 1680 problem instances in (the extended) Set1 and 1440 problem instances in (the extended) Set2. All problem instances can be obtained by requesting them via email from the first author of this paper.

4.2 Tuning of the Metaheuristics

The automatic configuration tool irace [15] was used for tuning the parameters of the three metaheuristics. The tuning processes are described in the following.

Tuning of Cmsa. The following parameters of Cmsa were considered for tuning: (n_a) the number of solution constructions per iteration, ($\text{age}_{\max}$) the maximum allowed age of solution components, (d_{rate}) the determinism rate, (l_{size}) the candidate list size, and ($t_{\max}$) the maximum time in seconds allowed for CPLEX per application to a sub-instance. In particular, Cmsa was tuned separately for each alphabet size, which—after initial experiments—seems to have a greater influence on the behavior of the algorithm than the length of the input strings. In the context of Set1, two tuning instances were randomly generated for each combination of string length and alphabet size, whereas for Set2 two tuning instances were randomly generated for each combination of alphabet size and number of repetitions.

The tuning process for each alphabet size was given a budget of 1000 runs of Cmsa, where each run was given a computation time limit of $l_x/10$ CPU seconds for instances of Set1 (remember that for instances of Set1 it holds that $l_x = l_y$) and $(|\Sigma| * reps)/10$ CPU seconds for instances of Set2. Finally, the following parameter value ranges were considered concerning the five parameters of Cmsa:

- $n_a \in \{10, 30, 50\}$

Table 2: Results of tuning Cmsa with irace.

$\|\Sigma\|$	n_a	age$_{\max}$	d_{rate}	l_{size}	$t_{\max}$
$n/8$	10	1	0.7	5	100.0
$n/4$	10	1	0.9	10	100.0
$3n/8$	10	1	0.7	10	5.0
$n/2$	10	1	0.5	3	100.0
$5n/8$	50	1	0.9	10	100.0
$3n/4$	10	1	0.5	5	10.0
$7n/8$	30	10	0.5	3	100.0

(a) Tuning results concerning Set1.

$\|\Sigma\|$	n_a	age$_{\max}$	d_{rate}	l_{size}	$t_{\max}$
4	30	5	0.5	3	1.0
8	30	*inf*	0.3	10	1.0
16	10	1	0.9	3	1.0
32	50	1	0.5	10	1.0
64	10	5	0.7	10	5.0
128	10	5	0.0	3	100.0
256	10	1	0.5	3	100.0
512	10	1	0.3	3	10.0

(b) Tuning results concerning Set2.

- age$_{\max}$ $\in \{1, 5, 10, \mathit{inf}\}$, where *inf* means that solution components are never removed from Z_{sub}.
- $d_{\text{rate}} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$, where a value of 0.0 means that the selection of the next variable to be added to the partial solution under construction is always done randomly from the candidate list, while a value of 0.9 means that solution constructions are nearly deterministic.
- $l_{\text{size}} \in \{3, 5, 10\}$
- $t_{\max} \in \{1.0, 5.0, 10.0, 100.0\}$ (in seconds).

The tuning runs with irace produced the configurations of Cmsa as shown in Table 2. At first sight, it might be surprising that no clear tendency about the setting of the parameter values can be observed. However, this can be explained by the fact that the algorithm is—at least for instances of small and medium size—very robust.

Tuning of Beam-Aco. The following parameters of Beam-Aco were considered for tuning: (k_{bw}) the beam width, that is, the number of partial solutions that can be maintained per construction step within beam search, (μ) a parameter used to determine the maximal number of solution extensions that may be chosen at each step, and (d_{rate}) the determinism rate. Note that the parameters of the ACO framework (such as the learning rate, etc.) were not found to have a big influence on the behavior of the algorithm, which is clearly dominated by the beam search component.

Table 3: Results of tuning BEAM-ACO with irace.

$\|\Sigma\|$	k_{bw}	μ	d_{rate}
$n/8$	10	1.5	0.3
$n/4$	30	2.5	0.3
$3n/8$	10	1.5	0.0
$n/2$	10	1.5	0.6
$5n/8$	10	1.5	0.0
$3n/4$	10	2.0	0.0
$7n/8$	10	1.5	0.6

(a) Tuning results concerning SET1.

$\|\Sigma\|$	k_{bw}	μ	d_{rate}
4	10	2.0	0.3
8	50	3.0	0.9
16	5	1.5	0.0
32	5	1.5	0.0
64	5	1.5	0.3
128	5	2.0	0.0
256	10	1.5	0.0
512	10	1.5	0.0

(b) Tuning results concerning SET2.

Just like in the case of CMSA, BEAM-ACO was tuned separately for each alphabet size. The tuning instances, the budget per tuning run, and the computation time limits were chosen as in the case of CMSA. The following parameter value ranges were considered concerning the three parameters of BEAM-ACO:

- $k_{\text{bw}} \in \{5, 10, 30, 50, 100\}$
- $\mu \in \{1.5, 2.0, 2.5, 3.0, 3.5\}$
- $d_{\text{rate}} \in \{0.0, 0.3, 0.6, 0.9\}$

The tuning runs with irace produced the configurations of BEAM-ACO as shown in Table 3. As in the case of CMSA, no clear tendency about the setting of the parameter values can be observed, which can be explained again by the remarkable robustness of the algorithm, at least in the context of instances of small and medium size.

Tuning of Ea. The three parameters of EA that were considered for tuning are the following ones: (p_{size}) the population size, (c_{rate}) the crossover rate, and (m_{rate}) the mutation rate. Just like in the cases of CMSA and BEAM-ACO, EA was tuned separately for each alphabet size. The tuning instances, the budget per tuning run, and the computation time limits were chosen as in the cases of CMSA and BEAM-ACO. The following parameter value ranges were considered concerning the three parameters of EA:

- $p_{\text{size}} \in \{10, 20, 40, 80, 160\}$.
- $c_{\text{rate}} \in \{0.75, 0.8, 0.85, 0.9, 0.95\}$.
- $m_{\text{rate}} \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$.

The tuning runs with irace produced the configurations of EA as shown in Table 4.

Table 4: Results of tuning EA with irace.

$\|\Sigma\|$	p_{size}	c_{rate}	m_{rate}
$n/8$	10	0.9	0.01
$n/4$	10	0.8	0.01
$3n/8$	10	0.85	0.03
$n/2$	10	0.8	0.01
$5n/8$	10	0.75	0.01
$3n/4$	10	0.95	0.02
$7n/8$	10	0.85	0.01

(a) Tuning results concerning SET1.

$\|\Sigma\|$	p_{size}	c_{rate}	m_{rate}
4	10	0.75	0.03
8	40	0.8	0.04
16	80	0.8	0.05
32	10	0.8	0.05
64	10	0.75	0.03
128	10	0.95	0.01
256	10	0.9	0.01
512	10	0.75	0.01

(b) Tuning results concerning SET2.

4.3 Statistical Assessment of the Results

As described in more detail below, the results (in terms of the obtained objective function values) are presented in each row of the result tables in terms of averages over 30 problem instances. The results obtained by the four considered methods concerning each table row were statistically tested in order to determine the significance of the differences among them. This was done by comparing the results of all algorithms with the result of the best-performing algorithm per table row. As an example, consider the 30 instances with $\Sigma = n/8$ and $n = 512$ (see Table 5). The best performing algorithm is CMSA, as marked by bold font. The results of BEAM-ACO are statistically equivalent, as indicated by the ★ symbol (significance level of 0.05). The differences have been assessed using Friedman's test, and the p-values have been corrected for multiple comparison using Finner's procedure [11].

Additionally, we aimed for detecting the differences between the algorithms (if any) for larger subsets of the considered instances. For doing so, all the algorithms have been compared simultaneously using Friedman's test. Then, given that in all the cases the test rejected the hypothesis that all the algorithms perform equally, all the pairwise comparisons have been performed using the Nemenyi post-hoc test [12]. The corresponding results will be shown in Section 4.4 by means of so-called criticial difference plots. Finally, note that all the tests and the plots have been generated using R's **scmamp** package, available at `https://github.com/b0rxa/scmamp`.

4.4 Numerical Results

As already mentioned before, four algorithmic techniques were included in the comparison: (1) CMSA (from [4]), (2) BEAM-ACO (from [5]), (3) EA from [9], and (4) CPLEX*, which refers to the application of CPLEX to all instances in relation to the ILP model from Section 2. The parameter values of the first three algorithms were chosen as described above. All four algorithmic techniques were applied to each problem instance with a computation time limit of $l_x/10$ CPU seconds for instances of SET1 (remember that for instances of SET1 it holds that $l_x = l_y$) and $(|\Sigma| * reps)/10$ CPU seconds for instances of SET2. Moreover, a memory limit of 4Gb was used for each application of Cplex*. However, this limit was never reached within the allowed computation time.

The numerical results are presented in Table 5 concerning Set1 and Table 6 concerning Set2. Each table row presents the results averaged over 30 problem instances of the same type. The results of Cmsa, Beam-Aco and Ea are provided in two columns each. The first one (with heading **result**) provides the result of the corresponding algorithm averaged over 30 problem instances, while the second column (with heading **time**) provides the average computation time (in seconds) necessary for finding the corresponding solutions. The result column is also provided for Cplex*. However, the second column for Cplex* provides the average optimality gaps (in percent), that is, the average gaps between the upper bounds and the values of the best solutions when stopping a run. Finally, the best result of each table row is indicated with bold font. Moreover, those algorithms whose performance is not significantly worse than the one of the best performing algorithm at a significance level of 0.05 are marked in the table with the ★ symbol. Those cases in which Cplex* was not able to produce a single feasible solution are marked with 'n.a.'.

The following observations can be made:

- As already noted in [4], Cplex* is very efficient in solving most of the problem instances of Set1 with $n \leq 256$ and of Set2 with $|\Sigma| \leq 32$. However, both concerning Set1 and Set2, there seems to be a sharp transition between instances that are easily solvable for Cplex* and those that are not. For example, for instances with $n \geq 1024$ (Set1) Cplex* is generally not able to come up even with a single feasible solution within the allowed computation time. The same holds for instance with $|\Sigma| \geq 256$ concerning Set2.

- In general, Ea is only competitive for the smallest problem instances in both problem instance sets. Moreover, the larger the tackled problem instances, the less competitive is Ea. Nevertheless, we would like to emphasize again that the results of our re-implementation of Ea are not worse than those of the original implementation from [9].

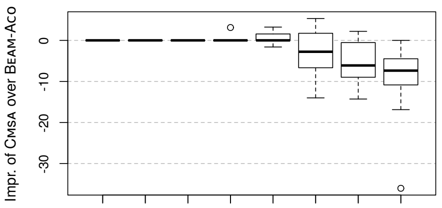
- Concerning Set1, both Cmsa and Beam-Aco provide (near-)optimal solutions and both outperform Cplex* once the average optimality gaps start to increase. Hereby, Cmsa is generally slightly better than Beam-Aco for instances with $n \leq 1024$. Starting from instances with $n = 2048$, this seems to change in the sense that, now, Beam-Aco has slight advantages over Cmsa. However, this is not surprising, as Cmsa is based on the application of Cplex* to reduced sub-instances and, even though applicable to larger problem instances than pure Cplex*, Cmsa was expected to reach its limits with growing problem instance size. Nevertheless, the performance differences between Cmsa and Beam-Aco are nearly always of no statistical significance; at least for what concerns the row-wise comparison of the results.

- Concerning Set2, the performance of Cmsa is again slightly better than the one of Beam-Aco for most instances with $|\Sigma| \leq 256$, providing (near-)optimal solutions. However, starting from alphabet size $|\Sigma| = 512$, Beam-Aco starts again perform slightly better than Cmsa, for the same reasons as outlined above.

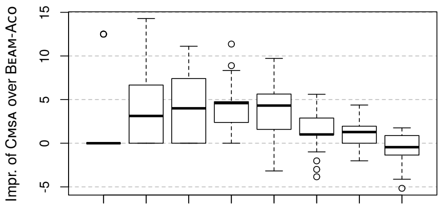
Apart from the numerical results provided in the form of tables, the graphics of Figure 1 and 2 show the relation between Cmsa and Beam-Aco in the following way. Figure 1 shows the improvement (in percent) of Cmsa over Beam-Aco in the context of Set1, and Figure 2 shows the same for Set2. Note that in the case of negative values, Beam-Aco performs better than Cmsa.

Given the general picture of Cmsa performing slightly better than Beam-Aco for small and medium size problem instances and of Beam-Aco performing slightly better than Cmsa in the context of the largest problem instances, we finally considered combining Cmsa and Beam-Aco in some way in order to obtain an algorithm that would perform well for all considered problem instances. In fact, apart from the restrictions implied by the application of CPLEX within Cmsa, it can also be assumed that, with growing problem instance size, the greedy construction mechanism of Cmsa is likely to be less able to feed the reduced problem instances with all components necessary in order to generate high-quality solutions. In turn, the construction mechanism of Beam-Aco—which, in addition to being based on the greedy function, is also strongly influenced by the upper bound function—seems to be able to generate high-quality solutions even in the context of large problem instances. Therefore, it was decided to test a version of Cmsa in which the initial reduced problem instance, in contrast to being empty, is composed of the solution components found in the best solution obtained by running the beam search algorithm from Algorithm 3 with the following parameter settings for all problem instances: $k_{\mathrm{b}w} = 30$, $\mu = 2.5$, and $d_{\mathrm{rate}} = 1.0$. That is, the beam search component is applied in a deterministic way—due to $d_{\mathrm{rate}} = 1.0$—and with an intermediate value of $k_{\mathrm{b}w} = 30$ in order not to use too much computation time. This version of Cmsa, henceforth referred to by Hyb-Cmsa, was applied with the same computation time limit as used for the other algorithms, to all problem instances. Moreover, the parameter value setting was the same as the one determined for Cmsa. The results are shown in Table 7 concerning Set1 and in Table 8 concerning Set2. In case Hyb-Cmsa obtains the best average value among all five approaches, the corresponding value is provided in bold font. If this is not the case, but the value obtained by Hyb-Aco is statistically equivalent to the one of the best-performing approache, this is marked by the ★ symbol. Finally, in those cases in which the result obtained by Hyb-Aco improves both over the one of Cmsa and the one of Beam-Aco, the corresponding table cells are painted with a lightgrey background.

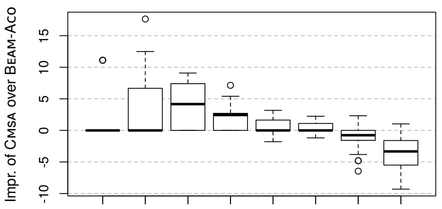
The following observation can be made. First, only in two cases the result of Hyb-Cmsa is slightly worse than the one of Cmsa. In turn, for most of the medium and large size problem instances, Hyb-Cmsa improves over both



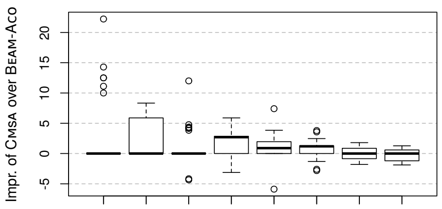
(a) Instances with $|\Sigma| = n/8$



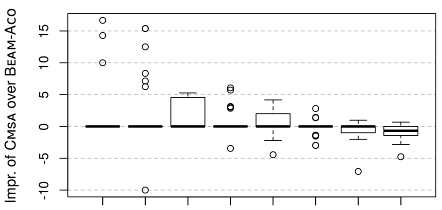
(b) Instances with $|\Sigma| = n/4$



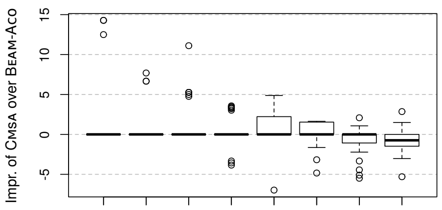
(c) Instances with $|\Sigma| = 3n/8$



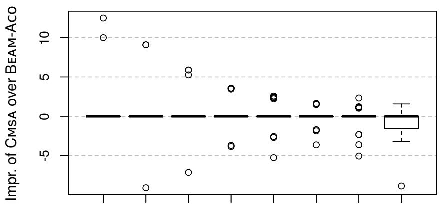
(d) Instances with $|\Sigma| = n/2$



(e) Instances with $|\Sigma| = 5n/8$



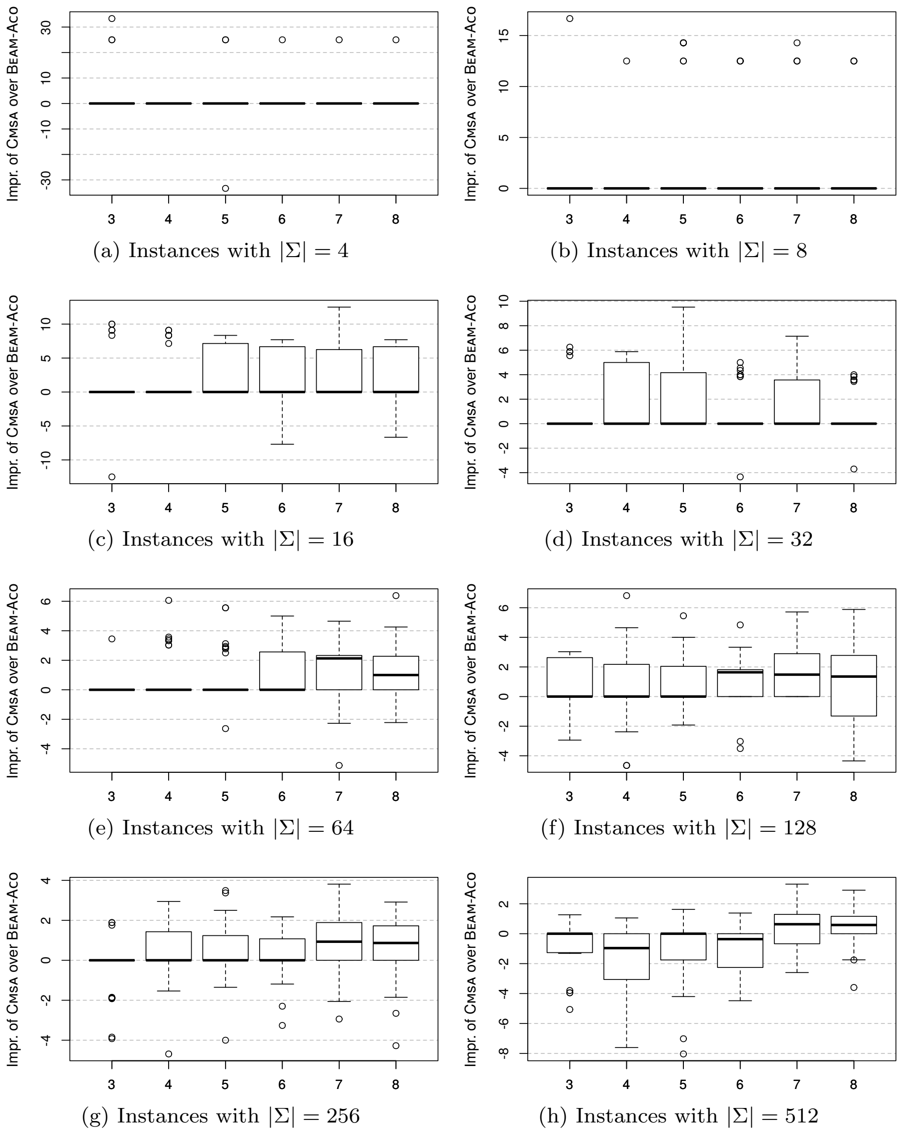
(f) Instances with $|\Sigma| = 3n/4$



(g) Instances with $|\Sigma| = 7n/8$

Figure 1: Improvement of Cmsa over Beam-Aco in the context of instances of Set1. Each box shows the differences between the objective function values of the solutions produced by Cmsa and the ones of the solutions produced by Beam-Aco for the 30 instances of the same type (in percent). The x-axis of each graphic ranges over the different string lengths.

Cmsa and Beam-Aco. This also happens frequently in cases in which Cmsa was already performing better than Cmsa. This clearly shows that Hyb-Cmsa



(a) Instances with $|\Sigma| = 4$

(b) Instances with $|\Sigma| = 8$

(c) Instances with $|\Sigma| = 16$

(d) Instances with $|\Sigma| = 32$

(e) Instances with $|\Sigma| = 64$

(f) Instances with $|\Sigma| = 128$

(g) Instances with $|\Sigma| = 256$

(h) Instances with $|\Sigma| = 512$

Figure 2: Improvement of Cmsa over Beam-Aco in the context of instances of Set2. Each box shows the differences between the objective function values of the solutions produced by Cmsa and the ones of the solutions produced by Beam-Aco for the 30 instances of the same type (in percent). The x-axis of each graphic ranges over the different values of *rep*.

benefits from synergies between Cmsa and Beam-Aco. Moreover, Hyb-Cmsa can be called a current state-of-the-art algorithm for the RFLCS problem.

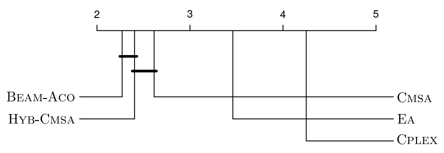
Additional statistical information comparing the four initially presented algorithms and the new hybrid algorithm is provided in terms of a comparison between the five algorithms for subsets of instances in Figure 3 (concerning SET1) and Figure 4 (concerning SET2). These figures present the result of the Nemenyi test in a graphical way. Briefly, each algorithm is positioned in the segment according to its average ranking concerning the considered subset of instances. Then, the critical difference (CD) is computed for a significance level of 0.05 and the performance of those algorithms that have a difference lower than CD are regarded as equal—that is, no difference of statistical significance can be detected. This is indicated in the graphic by horizontal lines joining the respective algorithms.
The following observations can be made:

- First, and most imortantly, when considering all instances of SET1, respectively SET2, then all algorithms are different with statistical significance (see Figures 3h and 4i). In both cases, the best-ranked algorithm is HYB-CMSA, followed by CMSA and BEAM-ACO (in this order).

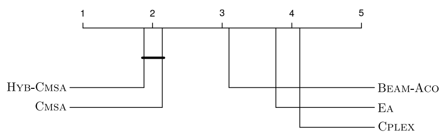
- Second, when considering subsets of instances, HYB-CMSA is also generally the best-ranked algorithm. However, not always with a statistically significant different. See, for example, the graphic in Figure 3e (concerning all instances of SET1 with $\Sigma = 5n/8$) where there is no statistical difference between HYB-CMSA and CMSA. Moreover, in some cases—when small problem instances are concerned—all five considered approaches are statistically equivalent; see, for example, the graphics in Figures 4a and 4b, concerning instances with $\Sigma = 4$ and $\Sigma = 8$.

5 Conclusions and Future Work

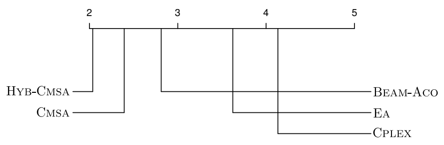
This work has provided a comprehensive comparison between the metaheuristic algorithms that were proposed for the repetition-free longest common subsequence problem in the literature. Moreover, the application of the general-purpose integer linear programming solver CPLEX was included in the comparison. The evaluation has shown that—using the specific ILP model outlined in this work—CPLEX is able to solve most of the problem instances so-far considered in the literature to optimality. Therefore, the experimental evaluation conducted in this work was extended to much larger problem instance than those considered so far. Based on the results of the different metaheuristics approaches, a hybrid approach combining the two best proposals from the literature, CMSA and Beam-ACO, was developed. The results have shown that this hybrid, indeed, benefits from the synergies between these two techniques, which makes the developed hybrid technique the current state-of-the-art algorithm for the repetition-free longest common subsequence problem. In particular, this was shown with statistical significance when considering all the instances of the first, respectively the second, benchmark set together.



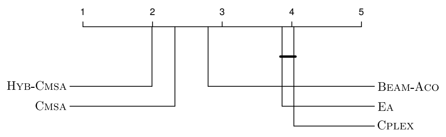
(a) Instances with $|\Sigma| = n/8$



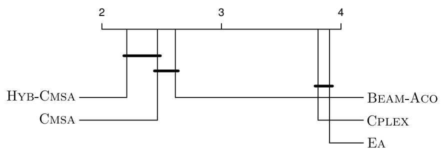
(b) Instances with $|\Sigma| = n/4$



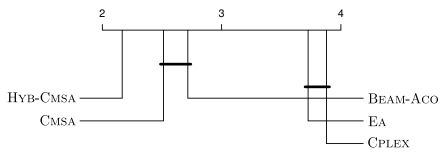
(c) Instances with $|\Sigma| = 3n/8$



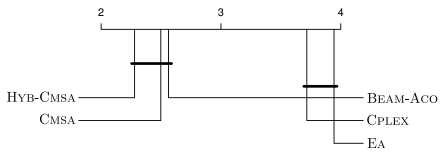
(d) Instances with $|\Sigma| = n/2$



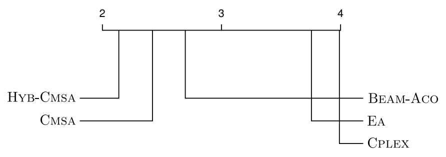
(e) Instances with $|\Sigma| = 5n/8$



(f) Instances with $|\Sigma| = 3n/4$



(g) Instances with $|\Sigma| = 7n/8$



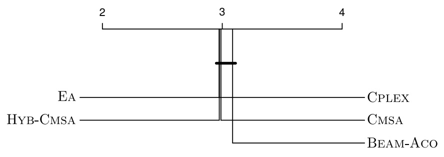
(h) All instances

Figure 3: Criticial difference plots for subsets of Set1 (see (a) to (g)), and globally for all instances of Set1(see (h)).

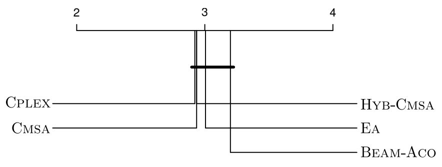
Concerning future work, we will consider different ways of hybridizing CMSA and Beam-ACO. Moreover, it might be interesting to experiment with the parallel use of different greedy functions in the context of CMSA.

Acknowledgements

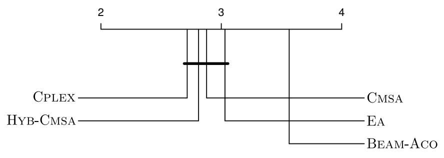
This work was supported by project TIN2012-37930-c02-02 (Spanish Ministry for Economy and Competitiveness, feder funds from the European Union). Maria J. Blesa acknowledges support by funds from the agaur of the Government of Catalonia under project ref. SGR 2014:1034 (albcom). Our experiments have been executed in the High Performance Computing environment managed by the rdlab of the Computer Science Department at the Technical University of Barcelona (`http://rdlab.cs.upc.edu`) and we would like to thank them for their support.



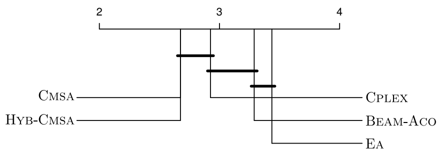
(a) Instances with $|\Sigma| = 4$



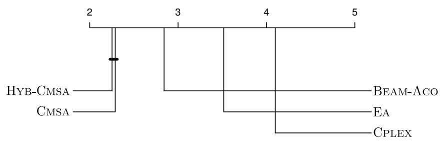
(b) Instances with $|\Sigma| = 8$



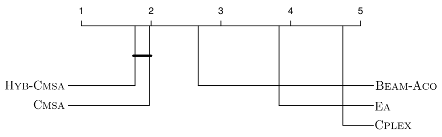
(c) Instances with $|\Sigma| = 16$



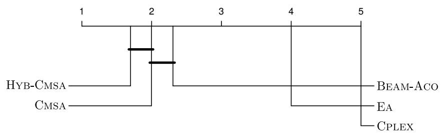
(d) Instances with $|\Sigma| = 32$



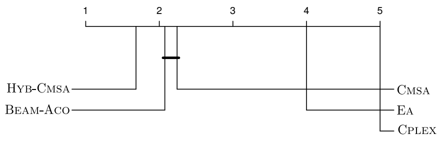
(e) Instances with $|\Sigma| = 64$



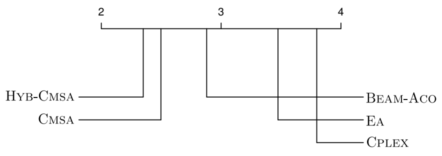
(f) Instances with $|\Sigma| = 128$



(g) Instances with $|\Sigma| = 256$



(h) Instances with $|\Sigma| = 512$



(i) All instances

Figure 4: Criticial difference plots for subsets of Set2 (see (a) to (h)), and globally for all instances of Set2(see (i)).

References

[1] Adi, S.S., Braga, M.D.V., Fernandes, C.G., Ferreira, C.E., Martinez, F.V., Sagot, M.F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsequence. Electronic Notes in Discrete Mathematics **30**, 243–248 (2008). Proceedings of The IV Latin-American Algorithms, Graphs, and Optimization Symposium

[2] Adi, S.S., Braga, M.D.V., Fernandes, C.G., Ferreira, C.E., Martinez,

F.V., Sagot, M.F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsquence. Discrete Applied Mathematics **158**, 1315–1324 (2010)

[3] Aho, A., Hopcroft, J., Ullman, J.: Data structures and algorithms. Addison-Wesley, Reading, MA (1983)

[4] Blum, C., Blesa, M.J.: Construct, merge, solve & adapt: Application to the repetition-free longest common subsequence problem. In: F. Chicano, B. Hu (eds.) Proceedings of EvoCOP 2016 – 16th European Conference on Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 9595, pp. 46–57. Springer Verlag, Berlin, Germany (2016)

[5] Blum, C., Blesa, M.J., Calvo, B.: Beam-ACO for the repetition-free longest common subsequence problem. In: P. Legrand, M.M. Corsini, J.K. Hao, N. Monmarché, E. Lutton, M. Schoenauer (eds.) Proceedings of EA 2013 – 11th Conference on Artificial Evolution, *Lecture Notes in Computer Science*, vol. 8752, pp. 79–90. Springer Verlag, Berlin, Germany (2014)

[6] Blum, C., Blesa, M.J., López-Ibáñez, M.: Beam search for the longest common subsequence problem. Computers & Operations Research **36**(12), 3178–3186 (2009)

[7] 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)

[8] Bonizzoni, P., Della Vedova, G., Dondi, R., Fertin, G., Rizzi, R., Vialette, S.: Exemplar longest common subsequence. IEEE/ACM Transactions on Computational Biology and Bioinformatics **4**(4), 535–543 (2007)

[9] Castelli, M., Beretta, S., Vanneschi, L.: A hybrid genetic algorithm for the repetition free longest common subsequence problem. Operations Research Letters **41**(6), 644–649 (2013)

[10] Easton, T., Singireddy, A.: A large neighborhood search heuristic for the longest common subsequence problem. Journal of Heuristics **14**(3), 271–283 (2008)

[11] García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information Sciences **180**(10), 2044 – 2064 (2010)

[12] García, S., Herrera, F.: An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of Machine Learning Research **9**, 2677 – 2694 (2008)

[13] Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)

[14] Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. Journal of Computational Biology **9**(2), 371–388 (2002)

[15] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles, Belgium (2011)

[16] Maier, D.: The complexity of some problems on subsequences and supersequences. Journal of the ACM **25**, 322–336 (1978)

[17] Ning, K.: Deposition and extension approach to find longest common subsequence for thousands of long sequences. Computational Biology and Chemistry **34**(3), 149–157 (2010)

[18] Smith, T., Waterman, M.: Identification of common molecular subsequences. Journal of Molecular Biology **147**(1), 195–197 (1981)

[19] Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology **147**(1), 195–197 (1981)

[20] Storer, J.: Data Compression: Methods and Theory. Computer Science Press, MD (1988)

[21] Tabataba, F.S., Mousavi, S.R.: A hyper-heuristic for the longest common subsequence problem. Computational Biology and Chemistry **36**, 42–54 (2012)

[22] Wang, Q., Korkin, D., Shang, Y.: A fast multiple longest common subsequence (MLCS) algorithm. IEEE Transactions on Knowledge and Data Engineering **23**(3), 321–334 (2011)

[23] Wang, Q., Pan, M., Shang, Y., Korkin, D.: A fast heuristic search algorithm for finding the longest common subsequence of multiple strings. In: Proceedings of AAAI – Conference on Artificial Intelligence, pp. 1287–1292 (2010)

Table 5: Experimental results (in terms of the average final objective function values) concerning the instances of Set1.

$\|\Sigma\|$	n	Cmsa		Beam-Aco		Ea		Cplex*		
		result	**time**	**result**	**time**	**result**	**time**	**result**	**time**	**gap**
$n/8$	32	**4.00**	< 1	**4.00**	< 1	**4.00**	< 1	**4.00**	< 1	0.0
	64	**8.00**	< 1	**8.00**	< 1	**8.00**	< 1	**8.00**	< 1	0.0
	128	**16.00**	< 1	**16.00**	< 1	**16.00**	< 1	**16.00**	7	0.0
	256	**31.97**	< 1	31.93★	< 1	31.83★	2	n.a.	n.a.	n.a.
	512	**63.27**	15	62.90★	7	58.33	33	n.a.	n.a.	n.a.
	1024	108.60★	72	**111.57**	26	86.00	87	n.a.	n.a.	n.a.
	2048	174.00★	160	**182.67**	102	107.10	173	n.a.	n.a.	n.a.
	4096	262.50	272	**283.33**	264	129.43	285	n.a.	n.a.	n.a.
$n/4$	32	**7.83**	< 1	7.70★	< 1	7.77★	< 1	**7.83**	< 1	0.0
	64	14.63★	< 1	14.07	< 1	14.07	< 1	**14.67**	< 1	0.0
	128	**25.70**	< 1	24.53	< 1	24.27	5	25.43★	9	5.3
	256	**43.70**	6	41.80	< 1	40.97	9	24.23	7	> 100
	512	**67.70**	9	65.17	< 1	59.63	36	n.a.	n.a.	n.a.
	1024	**102.53**	31	101.03★	< 1	73.47	58	n.a.	n.a.	n.a.
	2048	**153.63**	84	151.97★	1	91.97	151	n.a.	n.a.	n.a.
	4096	223.40★	234	**224.47**	7	117.57	163	n.a.	n.a.	n.a.
$3n/8$	32	**8.77**	< 1	8.67★	< 1	8.70★	< 1	**8.77**	< 1	0.0
	64	**15.53**	< 1	14.97	< 1	15.33★	< 1	**15.53**	< 1	0.0
	128	24.87★	< 1	23.90	< 1	24.37	2	**24.90**	3	0.0
	256	**39.93**	< 1	38.97	< 1	38.20	11	21.80	8	> 100
	512	**59.60**	6	59.37★	4	48.77	29	5.5	37	> 100
	1024	**90.27**	19	89.87★	16	63.83	54	n.a.	n.a.	n.a.
	2048	129.00★	58	**130.07**	34	79.43	79	n.a.	n.a.	n.a.
	4096	184.57	385	**190.87**	100	107.67	91	n.a.	n.a.	n.a.
$n/2$	32	**8.87**	< 1	8.63★	< 1	8.53★	< 1	**8.87**	< 1	0.0
	64	14.77★	< 1	14.50★	< 1	14.10	< 1	**14.80**	< 1	0.0
	128	22.83★	< 1	22.60★	< 1	22.00	2	**22.93**	1	0.0
	256	**34.97**	< 1	34.33★	< 1	33.10	9	29.00	18	62.6
	512	**52.83**	6	52.13★	< 1	46.50	20	10.70	28	> 100
	1024	**78.50**	21	78.00★	1	61.30	58	n.a.	n.a.	n.a.
	2048	**114.70**	25	114.57★	6	76.87	111	n.a.	n.a.	n.a.
	4096	166.07★	106	**166.33**	25	102.10	136	n.a.	n.a.	n.a.
$5n/8$	32	8.57★	< 1	8.47★	< 1	8.23★	< 1	**8.60**	< 1	0.0
	64	13.27★	< 1	13.00★	< 1	12.40	< 1	**13.30**	< 1	0.0
	128	**21.20**	< 1	20.83★	< 1	20.33	3	**21.20**	< 1	0.0
	256	32.47★	< 1	32.20★	< 1	31.33	9	**32.53**	10	0.0
	512	**47.60**	1	47.40★	< 1	43.47	23	25.23	19	> 100
	1024	69.87★	10	**70.00**	8	56.00	57	n.a.	n.a.	n.a.
	2048	103.07★	44	**103.63**	15	72.83	129	n.a.	n.a.	n.a.
	4096	148.17★	200	**149.57**	75	96.43	208	n.a.	n.a.	n.a.
$3n/4$	32	**8.17**	< 1	8.07★	< 1	7.93★	< 1	**8.17**	< 1	0.0
	64	**12.53**	< 1	12.43★	< 1	12.37★	< 1	**12.53**	< 1	0.0
	128	**19.70**	< 1	19.47★	< 1	19.53★	2	**19.70**	< 1	0.0
	256	29.87★	< 1	29.80★	< 1	29.00	7	**29.97**	5	0.0
	512	**44.30**	2	43.97★	< 1	39.47	17	21.97	14	> 100
	1024	**64.63**	3	64.40★	< 1	51.90	56	0.03	75	> 100
	2048	93.13★	30	**93.80**	17	67.40	125	n.a.	n.a.	n.a.
	4096	134.67★	71	**135.50**	20	90.70	213	n.a.	n.a.	n.a.
$7n/8$	32	**7.67**	< 1	7.60★	< 1	7.53★	< 1	**7.67**	< 1	0.0
	64	11.53★	< 1	11.50★	< 1	10.97	< 1	**11.57**	< 1	0.0
	128	18.37★	< 1	18.30★	< 1	17.70	2	**18.40**	< 1	0.0
	256	27.73★	< 1	27.63★	< 1	26.93	7	**27.80**	2	0.0
	512	**40.43**	2	40.37★	< 1	37.57	25	33.13	34	84.9
	1024	60.30★	13	**60.37**	3	50.30	48	1.17	50	> 100
	2048	87.60★	43	**87.70**	2	65.83	123	n.a.	n.a.	n.a.
	4096	125.60★	107	**126.53**	20	86.03	186	n.a.	n.a.	n.a.

Table 6: Experimental results (in terms of the average final objective function values) concerning the instances of Set2.

$\|\Sigma\|$	*reps*	Cmsa		Beam-Aco		Ea		Cplex*		
		result	**time**	**result**	**time**	**result**	**time**	**result**	**time**	**gap**
4	3	**3.47**	< 1	3.37★	< 1	**3.47**	< 1	**3.47**	< 1	0.0
	4	**3.77**	< 1	**3.77**	< 1	**3.77**	< 1	**3.77**	< 1	0.0
	5	3.80★	< 1	3.77★	< 1	**3.83**	< 1	**3.83**	< 1	0.0
	6	**3.90**	< 1	3.87★	< 1	**3.90**	< 1	**3.90**	< 1	0.0
	7	**3.97**	< 1	3.93★	< 1	**3.97**	< 1	**3.97**	< 1	0.0
	8	**3.97**	< 1	3.93★	< 1	**3.97**	< 1	**3.97**	< 1	0.0
8	3	**6.23**	< 1	6.20★	< 1	**6.23**	< 1	**6.23**	< 1	0.0
	4	6.83★	< 1	6.80★	< 1	6.83★	< 1	**6.87**	< 1	0.0
	5	**7.40**	< 1	7.20★	< 1	7.33★	< 1	**7.40**	< 1	0.0
	6	**7.53**	< 1	7.40★	< 1	7.50★	< 1	**7.53**	< 1	0.0
	7	**7.70**	< 1	7.57★	< 1	7.63★	< 1	**7.70**	< 1	0.0
	8	**7.77**	< 1	7.67★	< 1	**7.77**	< 1	**7.77**	< 1	0.0
16	3	9.67★	< 1	9.53★	< 1	9.60★	< 1	**9.70**	< 1	0.0
	4	**11.57**	< 1	11.37★	< 1	11.47★	< 1	**11.57**	< 1	0.0
	5	**12.93**	< 1	12.57★	< 1	12.80★	< 1	**12.93**	< 1	0.0
	6	13.83★	< 1	13.57	< 1	13.93★	< 1	**14.00**	< 1	0.0
	7	14.87★	< 1	14.50★	< 1	14.80★	< 1	**14.93**	< 1	0.0
	8	14.67★	< 1	14.30	< 1	14.57★	1	**14.80**	< 1	0.0
32	3	**16.13**	< 1	15.97★	< 1	16.03★	< 1	**16.13**	< 1	0.0
	4	**19.00**	< 1	18.67★	< 1	18.80★	2	**19.00**	< 1	0.0
	5	**21.63**	< 1	21.23★	< 1	21.23★	2	**21.63**	1	0.0
	6	23.70★	< 1	23.50★	< 1	23.37★	2	**23.73**	4	0.0
	7	**25.53**	< 1	25.20★	1	24.83	2	25.13★	9	2.9
	8	**27.40**	1	27.23★	1	26.63	2	25.77	16	12.2
64	3	**25.43**	< 1	25.40★	< 1	25.10★	3	**25.43**	< 1	0.0
	4	**30.37**	< 1	30.10★	< 1	30.00★	9	**30.37**	7	0.0
	5	**34.83**	1	34.57★	1	34.30★	9	31.50	23	32.4
	6	**39.03**	6	38.50★	5	37.73	17	24.13	9	> 100
	7	**43.40**	9	42.90★	3	41.23	20	24.60	9	> 100
	8	**45.17**	16	44.60★	6	42.23	23	23.30	10	> 100
128	3	**36.47**	2	36.17★	< 1	36.00★	12	33.23	29	35.9
	4	**44.63**	3	44.23★	5	41.83	17	20.63	13	> 100
	5	**53.07**	9	52.43★	7	48.90	26	20.87	25	> 100
	6	**60.90**	15	60.10★	8	55.10	47	13.87	51	> 100
	7	**67.73**	27	66.50★	12	60.80	48	n.a.	n.a.	n.a.
	8	**73.13**	37	72.27★	20	66.00	71	n.a.	n.a.	n.a.
256	3	54.70★	3	**54.87**	< 1	49.87	30	4.23	25	> 100
	4	**68.47**	9	68.30★	4	57.27	41	n.a.	n.a.	n.a.
	5	**80.77**	20	80.33★	7	63.10	58	n.a.	n.a.	n.a.
	6	**92.60**	31	92.37★	34	70.30	79	n.a.	n.a.	n.a.
	7	**102.93**	28	102.30★	35	75.67	114	n.a.	n.a.	n.a.
	8	**113.30**	46	112.77★	37	81.20	140	n.a.	n.a.	n.a.
512	3	80.63★	17	**81.27**	10	66.17	76	n.a.	n.a.	n.a.
	4	99.13★	17	**100.63**	27	74.57	107	n.a.	n.a.	n.a.
	5	118.90★	63	**120.20**	63	80.47	124	n.a.	n.a.	n.a.
	6	135.57★	63	**136.73**	62	86.73	164	n.a.	n.a.	n.a.
	7	**153.57**	108	153.23★	112	92.73	197	n.a.	n.a.	n.a.
	8	**172.00**	124	171.30★	102	100.23	249	n.a.	n.a.	n.a.

Table 7: Experimental results (in terms of the average final objective function values) of Hyb-Cmsa concerning the instances of Set1.

n \ $\lvert\Sigma\rvert$	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	**4.00**	**7.83**	**8.77**	**8.87**	8.57★	**8.17**	**7.67**
64	**8.00**	14.63★	**15.53**	14.77★	**13.30**	**12.53**	**11.57**
128	**16.00**	**25.77**	24.87★	22.90★	**21.20**	**19.70**	**18.40**
256	**31.97**	43.63★	**39.97**	**35.10**	32.50★	**29.97**	**27.80**
512	63.13★	**67.90**	**59.77**	**53.10**	**47.83**	**44.53**	**40.57**
1024	111.17★	**103.00**	**90.50**	**79.03**	**70.03**	**65.07**	**60.50**
2048	180.50★	**154.33**	**130.57**	**115.30**	**103.80**	**94.53**	**88.00**
4096	279.17★	**226.67**	**191.37**	**167.47**	**150.00**	**136.57**	**127.20**

Table 8: Experimental results (in terms of the average final objective function values) of Hyb-Cmsa concerning the instances of Set2.

reps \ $\lvert\Sigma\rvert$	4	8	16	32	64	128	256	512
3	**3.47**	**6.23**	**9.70**	**16.13**	**25.43**	**36.70**	**54.97**	**81.57**
4	**3.77**	6.83★	**11.57**	**19.00**	**30.37**	**44.90**	**68.70**	**100.83**
5	**3.83**	**7.40**	**12.93**	**21.63**	**34.87**	**53.23**	**81.00**	**120.43**
6	**3.90**	**7.53**	13.87★	23.70★	**39.07**	**61.07**	**93.10**	**137.03**
7	**3.97**	**7.70**	14.87★	**25.53**	**43.50**	**67.90**	**103.50**	**154.57**
8	**3.97**	**7.77**	14.77★	**27.40**	**45.17**	**73.57**	**113.70**	**172.10**