Large Neighborhood Search for the Most Strings With Few Bad Columns Problem *

Evelia Lizárraga[1], Maria J. Blesa[1], Christian Blum[2], and Günther R. Raidl[3]

[1]Computer Science Department, Universitat Politécnica de Catalunya – BarcelonaTech, Barcelona (Spain), {mjblesa,evelial}@cs.upc.edu

[2]Artificial Intelligence Research Institute, Spanish National Research Council (IIIA-CSIC), Bellaterra (Spain), christian.blum@iiia.csic.es

[3]Institute of Computer Graphics and Algorithms, Technische Universität Wien, Vienna (Austria), raidl@ac.tuwien.ac.at

Abstract

In this work we consider the following NP-hard combinatorial optimization problem from computational biology. Given a set of input strings of equal length, the goal is to identify a maximum cardinality subset of strings that differ maximally in a pre-defined number of positions. First of all we introduce an integer linear programming model for this problem. Second, two variants of a rather simple greedy strategy are proposed. Finally, a large neighborhood search algorithm is presented. A comprehensive experimental comparison among the proposed techniques shows, first, that larger neighborhood search generally outperforms both greedy strategies. Second, while large neighborhood search shows to be competitive with the stand-alone application of CPLEX for small and medium sized problem instances, it outperforms CPLEX in the context of larger instances.

Keywords: Most strings with few bad columns, integer linear programming, large neighborhood search.

*A preliminary version of this work appeared at the IEEE 2015 International Symposium on INnovations in Intelligent SysTems and Applications (INISTA), September 2-4, 2015, Madrid, Spain. This work was supported by project TIN2012-37930-C02-02 (Spanish Ministry for Economy and Competitiveness, FEDER funds from the European Union) and project SGR 2014-1034 (AGAUR, Generalitat de Catalunya). Additionally, Christian Blum acknowledges support from IKERBASQUE. Evelia Lizárraga acknowledges support from the Mexican National Council for Science and Technology (CONACYT, doctoral grant number 253787).

1 Introduction

Many optimization problems from the field of computational biology are concerned with strings such as, for example, DNA or protein sequences. Examples of such optimization problems dealing with strings include the longest common subsequence problem and its variants [3, 13], string selection problems [7, 8, 9], alignment problems [2, 12], and similarity search [11]. In this article we consider an NP-hard optimization problem—the so-called *most strings with few bad columns* (MSFBC) problem—which was introduced in [1] in order to model the following situation. Suppose to be given a set of DNA sequences from a heterogeneous population consisting of two subgroups: (1) a large subset of DNA sequences that are identical apart from at most k positions at which mutations may have occurred, and (2) a subset of outliers. The goal of the MSFBC problem is to separate the two subsets.

The MSFBC problem can technically be described as follows. Given is a set I of n input strings of length m over a finite alphabet Σ, that is, $I = \{s_1, \ldots, s_n\}$. The j-th position of a string s_i, $i = 1, \ldots, n$, $j = 1, \ldots, m$, is henceforth denoted by $s_i[j]$. Moreover, given is a fixed value $k < m$. Set I together with value k define a problem instance (I, k). The goal is to find a subset $S \subseteq I$ of maximal size such that the strings in S differ in at most k positions. In this context, the strings of a subset $S \subseteq I$ are said to differ in a position $1 \leq j \leq m$ if, and only if, at least two strings $s_i, s_r \in S$ exist such that $s_i[j] \neq s_r[j]$. A position j in which the strings from S differ is also called a *bad column*. This notation is derived from the fact that the set of input strings can conveniently be seen in form of a matrix in which the strings are the rows.

1.1 Existing Work

The authors of [1] showed that no polynomial-time approximation scheme (PTAS) for the MSFBC problem exists. Moreover, they state that the problem is a generalization of the problem of finding tandem repeats in a string [4]. In a preliminary version of this article (see [5]) we introduced the first integer linear programming (ILP) model for the MSFBC problem. Moreover, we devised a greedy strategy which was further applied in the context of a so-called pilot method [14]. For problem instances of small and medium size the best results were obtained by solving the ILP model by the general-purpose ILP solver CPLEX[1], while the greedy-based pilot method scaled much better to large problem instances. On the downside, also the pilot method consumed a rather large amount of computation time.

1.2 Our Contribution

In this work we, first, introduce an alternative way of making use of the greedy strategy that was introduced in [5]. This new way of using the greedy strategy is shown to obtain results comparable to the ones obtained by the greedy-based pilot method in significantly less computation time. Second, we introduce an ILP-based large neighborhood search (LNS) algorithm for the tackled problem. In general, LNS algorithms are based on the following idea (for more details see, for example, [10]). Given a valid solution to the tackled problem instance—also

[1] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`

called the incumbent solution—first, destroy selected parts of it, resulting in a partial solution. Then apply some other, possibly exact, technique to find a best valid solution on the basis of the given partial solution, that is, a best valid solution that contains the given partial solution. Thus, the destroy-step defines a *large neighborhood*, from which a best (or nearly best) solution is determined not by naive enumeration but by the application of a more effective alternative technique. In the case of the MSFBC problem we make use of our ILP model and the ILP-solver CPLEX for exploring large neighborhoods. Therefore, the LNS approach proposed in this paper can be labelled an ILP-based LNS. The results show that this ILP-based LNS approach is competitive with the stand-alone application of CPLEX for small and medium sized problem instances, and it outperforms CPLEX in the context of larger instances.

1.3 Organization of this Paper

The remainder of this paper is organized as follows. Section 2 introduces the ILP model for the MSFBC problem, while Section 3 describe the greedy strategies. Finally, Section 4 presents the ILP-based LNS approach. An experimental comparison is performed in Section 5, and Section 6 is dedicated to conclusions and an outlook to future work.

2 An ILP Model for the MSFBC Problem

For the description of the ILP model let $\Sigma_j \subseteq \Sigma$ be the set of letters appearing at least once at the j-th position of the n input strings. In technical terms, $\Sigma_j := \{s_i[j] \mid i = 1, \ldots, n\}$. The proposed ILP model for the MSFBC problem makes use of several types of binary incidence variables. First, for each input string s_i there is a binary variable x_i. In case $x_i = 1$, the corresponding input string s_i is part of the solution, i.e., $s_i \in S$, otherwise not. Furthermore, for each combination of a position $j = 1, \ldots, m$ and a letter $a \in \Sigma_j$ the model makes use of a binary variable z_j^a, which is forced—by means of adequate inequalities—to assume value one ($z_j^a = 1$) in case at least one string s_i with $s_i[j] = a$ is chosen for the solution. Finally, there is a binary variable y_j for each position $j = 1, \ldots, m$. Variable y_j is forced to assume value one ($y_j = 1$) in case the strings chosen for the solution differ at position j. Given these variables, the ILP can be formulated as follows.

$$\max \quad \sum_{i=1}^{n} x_i \tag{1}$$

$$\text{s.t.} \quad x_i \leq z_j^{s_i[j]} \qquad \text{for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \tag{2}$$

$$\sum_{a \in \Sigma_j} z_j^a \leq 1 + (|\Sigma_j| - 1) \cdot y_j \qquad \text{for } j = 1, \ldots, m \tag{3}$$

$$\sum_{j=1}^{m} y_j \leq k \tag{4}$$

$$x_i \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n \tag{5}$$

$$z_j^a \in \{0, 1\} \qquad \text{for } j = 1, \ldots, m \text{ and } a \in \Sigma_j \tag{6}$$

$$y_j \in \{0, 1\} \qquad \text{for } j = 1, \ldots, m \tag{7}$$

The objective function (1) maximizes the number of chosen strings. Equations (2) ensure that, if a string s_i is selected ($x_i = 1$), the variable $z_j^{s_i[j]}$, which indicates that letter $s_i[j]$ appears at position j in at least one of the selected strings, has value one. Furthermore, equations (3) ensure that y_j is set to one in case the selected strings differ at position j. Note that, in comparison to the original ILP model as described in [], the right hand side of (3) was strengthened from $1 + |\Sigma_j| \cdot y_j$ to $1 + (|\Sigma_j| - 1) \cdot y_j$. Finally, constraint (4) ensures that not more than k bad columns are permitted.

3 Heuristic Approaches

In [5] we introduced the following greedy strategy for completing a solution when given a non-empty partial solution—that is, a subset of the input strings $S^p \subseteq I$—as input. Henceforth, $\text{bc}(S^p)$ denotes the number of bad columns with respect to S^p, that is, the number of columns j such that at least two strings $s_i, s_r \in S^p$ exist with $s_i[j] \neq s_r[j]$. A valid partial solution S^p to the MSWBC problem fulfills the following two conditions:

1. $\text{bc}(S^p) \leq k$
2. There exists at least one string $s_l \in I \setminus S^p$ such that $\text{bc}(S^p \cup \{s_l\}) \leq k$.

Obviously, a valid complete solution S only fulfills the first one of these conditions.

The procedure for completing a given partial solution, henceforth denoted by $\mathsf{MakeSolutionComplete}(S^p)$, is shown in pseudo-code in Algorithm 1. It works as follows. At each iteration, exactly one of the strings from $I \setminus S^p$ is chosen, according to a weighting function, and added to S^p. The weighting function that is used concerns the number of bad columns. More specifically, among all strings from $E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$ one for which $\text{bc}(S^p \cup \{s\})$ is minimal is selected. In other words, at each iteration a string that causes a minimal increase in the number of bad columns is chosen. In case of ties, the first string encountered is selected. Note, in this context, that in our implementation the values of the weighting function bc() are calculated in an efficient way, that is,

Algorithm 1 Procedure MakeSolutionComplete(S^p)

input: a problem instance (I, k), a non-empty partial solution S^p
$E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$
while $E \neq \emptyset$ **do**
  $s^* := \text{argmin}\{\text{bc}(S^p \cup \{s\}) \mid s \in E\}$
  $S^p := S^p \cup \{s^*\}$
  $E := \{s \in E \mid \text{bc}(S^p \cup \{s\}) \leq k\}$
end while
output: A complete solution $S = S^p$

the bad columns concerning a partial solution S^p are stored and for calculating $\text{bc}(S^p \cup s)$ for a string $s \in I \setminus S^p$ only the remaining (non-bad) columns are considered.

3.1 Frequency-based Greedy

In [5] we introduced two different ways of using procedure MakeSolutionComplete(S^p) when starting with an empty solution. In the first one, exactly one string from I is chosen based on a criterion related to letter frequencies. This results in a partial solution containing exactly one string. MakeSolutionComplete(S^p) is then applied to this partial solution in order to obtain a complete solution.

The way in which the first string is chosen works as follows. Let $\text{fr}_{j,a}$ for all $a \in \Sigma$ and $j = 1, \ldots, m$ be the frequency of letter a at position j in the input strings from I. For example, if a appears in five of the n input strings at position j, $\text{fr}_{j,a} = 5/n$. With this definition, the following measure can be computed for each $s_i \in I$:

$$\omega(s_i) := \sum_{j=1}^{m} \text{fr}_{j,s_i[j]} \tag{8}$$

Remember, in this context, that $s_i[j]$ denotes the letter at position j of string s_i. In words, $\omega(s_i)$ is calculated as the sum of the frequencies of the letters at each position in s_i. The following string is then chosen to form the initial partial solution for MakeSolutionComplete(S^p):

$$s := \text{argmax}\{\omega(s_i) \mid s_i \in I\} \tag{9}$$

The advantages of this greedy algorithm—henceforth denoted by Greedy-F—are its simplicity and low resource requirements.

3.2 Multi-Start Greedy Approach

Apart from Greedy-F, in [5] we made use of procedure MakeSolutionComplete(S^p) in the context of a pilot method [14]. Procedure MakeSolutionComplete(S^p) was used at each level of the search tree in order to evaluate each possible partial solution. The choice of a partial solution at each level was then made on the basis of these evaluations.

The disadvantage of this pilot method was the excessive computation time consumption. With the aim of devising an algorithm yielding solutions with

Algorithm 2 Multi-start greedy method

1: **input:** a problem instance (I, k)
2: $S_{\text{bsf}} := \emptyset$
3: **for** $i = 1, \ldots, n$ **do**
4: $\quad S^p := \{s_i\}$
5: $\quad S :=$ MakeSolutionComplete(S^p)
6: $\quad$ **if** $|S| > |S_{\text{bsf}}|$ **then** $S_{\text{bsf}} := S$ **end if**
7: **end for**
8: **output:** Solution S_{bsf}

a quality comparable to the ones of the pilot method, we studied truncated versions of the pilot method. The one with the best balance between solution quality and computation time is shown in 2, and is best described as a *multi-start greedy approach*. It works as follows. For each possible string of $s_i \in I$ the corresponding partial solution $S^p = \{s_i\}$ is fed to procedure MakeSolutionComplete(S^p). The output of this multi-start greedy approach, henceforth called Greedy-Ms is the best one of the n solutions constructed in this way.

4 ILP-based Large Neighborhood Search

In the preliminary study [5] we applied CPLEX to the ILP (1)–(7) from Section 2 for a range of different MSFBC problem instances. While CPLEX performed quite well for small and medium sized instances, it failed even to provide nearly-optimal solutions in the context of large instances. In order to still be able to profit from CPLEX when dealing with large instances, we consider in the following an ILP-based LNS approach.

At each iteration, first, the current solution is destroyed partially. This can be done randomly or according to some heuristic criterion, resulting in a partial solution. Then, CPLEX is applied to find the best valid solution that contains the given partial solution obtained in the destruction step. Note that, given a partial solution S^p, the best complete solution containing S^p can be found by applying CPLEX to the ILP (1)–(7), augmented by the following variable fixations:

$$x_i = 1 \qquad \text{for } s_i \in S^p \tag{10}$$

The framework for the ILP-based LNS algorithm is shown in Algorithm 3. In line 2, heuristic Greedy-F—as described in Section 3.1—is applied to the tackled problem instance (I, k) in order to generate the first incumbent solution S_{bsf}. Then, at each iteration, the current incumbent solution S_{bsf} is partially destroyed (see line 5) by removing a certain percentage $\mathbf{perc}_{\text{dest}}$ of the strings in S_{bsf}. This results in a partial solution S^p. After initial experiments we decided to choose the strings to be removed uniformly at random. Given $\mathbf{perc}_{\text{dest}}$, note that the precise number d of strings to be removed is as follows:

$$d := \left\lfloor \frac{\mathbf{perc}_{\text{dest}} \cdot |S_{cur}|}{100} \right\rfloor \tag{11}$$

The next step consists in applying the ILP solver CPLEX in order to find a best solution S'_{opt} that contains the partial solution S^p (see line 6). In order to avoid

Algorithm 3 ILP-based LNS for the MSFBC problem

1: **input:** a problem instance (I, k), values for the algorithm parameters
2: S_{bsf} := Result of applying GREEDY-F
3: $\mathbf{perc}_{\text{dest}} := \mathbf{perc}^{\text{l}}_{\text{dest}}$
4: **while** CPU time limit not reached **do**
5: S^p := DestroyPartially$(S_{bsf}, \mathbf{perc}_{\text{dest}})$
6: S'_{opt} := ApplyILPSolver$(S^p, t_{\max})$
7: **if** $|S'_{\text{opt}}| > |S_{\text{bsf}}|$ **then**
8: $S_{\text{bsf}} := S'_{\text{opt}}$
9: $\mathbf{perc}_{\text{dest}} := \mathbf{perc}^{\text{l}}_{\text{dest}}$
10: **else**
11: $\mathbf{perc}_{\text{dest}} := \mathbf{perc}_{\text{dest}} + 5$
12: **if** $\mathbf{perc}_{\text{dest}} > \mathbf{perc}^{\text{u}}_{\text{dest}}$ **then**
13: $\mathbf{perc}_{\text{dest}} := \mathbf{perc}^{\text{l}}_{\text{dest}}$
14: **end if**
15: **end if**
16: **end while**
17: **output:** S_{bsf}

that this step takes too much computation time, CPLEX is given a time limit $t_{\max}$. The output of CPLEX is, therefore, a best—possibly optimal—solution found within the allowed computation time.

We decided to give the algorithm the possibility to dynamically adapt the percentage $\mathbf{perc}_{\text{dest}}$ of strings to be removed from the current solution. In general, setting the percentage of destruction to a suitable value is important to have a reasonable chance of escaping poor local optima but to nevertheless lose not too much information of an already obtained high quality solution. In order to control a dynamically changing value of $\mathbf{perc}_{\text{dest}}$ lower and upper bounds $0 \leq \mathbf{perc}^{\text{l}}_{\text{dest}} \leq \mathbf{perc}^{\text{u}}_{\text{dest}} \leq 100$ are specified as strategy parameters, and $\mathbf{perc}_{\text{dest}}$ will always have a value within these limits.

Initially, $\mathbf{perc}_{\text{dest}}$ is set to the lower bound (see line 3 of Algorithm 3). Then, at each iteration, if S'_{opt} is better than S_{bsf}, the value of $\mathbf{perc}_{\text{dest}}$ is set back to the lower bound $\mathbf{perc}^{\text{l}}_{\text{dest}}$. Otherwise, $\mathbf{perc}_{\text{dest}}$ is incremented by a certain amount. Following preliminary experiments, we set this amount to five. If $\mathbf{perc}_{\text{dest}}$ exceeds the upper bound $\mathbf{perc}^{\text{u}}_{\text{dest}}$, it is reset to the lower bound $\mathbf{perc}^{\text{l}}_{\text{dest}}$. These adaption steps are realized in Algorithm 3 in lines 7–15.

Note that the idea behind this way of dynamically changing the value of $\mathbf{perc}_{\text{dest}}$ is as follows. As long as the algorithm is able to improve the current solution S_{bsf} using a low destruction percentage, this percentage is kept low. In this way, the size of the large neighborhood is rather low and CPLEX is faster in deriving the corresponding optimal solutions. Only when the algorithm appears not to be able to improve over the current solution S_{bsf}, the destruction percentage is increased in a step-wise way in order to diversify the search.

5 Experiments

In this section we present a comprehensive experimental evaluation of the following four algorithmic approaches: (1) The frequency-based greedy method

GREEDY-F from Section 3.1, (2) the multi-start greedy method GREEDY-MS from Section 3.2 (3), the LP-based large neighborhood search approach LNS from Section 4, and (4) the application of CPLEX to the ILP model from Section 2, denoted by CPLEX. All approaches were implemented in ANSI C++ using GCC 4.7.3 for compiling the software. CPLEX was used in version 12.1 in single-threaded execution. The experimental results that are presented in the following were obtained on a cluster of computers with Intel® Xeon® CPU 5670 CPUs of 12 nuclei of 2933 MHz and (in total) 32 Gigabytes of RAM. For each run of CPLEX we allowed a maximum of 4 Gigabytes of RAM. In the following we first describe the set of benchmark instances. Then we present a detailed analysis of the experimental results.

5.1 Benchmark Instances

For the experimental comparison of the methods considered in this work we generated a set of random instances. These random instances are characterized by four different parameters: (1) the number of input strings n, (2) the length of the input strings m, (3) the alphabet size $|\Sigma|$, and (4) the so-called *change probability* $\mathbf{p}_c$. The generation of a random instance works as follows. First, a base string s of length m is generated uniformly at random, that is, each letter $a \in \Sigma$ has a probability of $1/|\Sigma|$ to appear at any of the m positions of s. Then, each of the n input strings of the problem instance under construction is derived as follows. First, s is copied into a new string s'. Then, each letter of s' is exchanged for a randomly chosen letter from Σ with a probability of $\mathbf{p}_c$. Note that the new letter is not necessarily different from the original one. Moreover, note that we enforced at least one change per input string.

The following values were used for the generation of the benchmark set:

- $n \in \{100, 500, 1000\}$
- $m \in \{100, 500, 1000\}$
- $|\Sigma| \in \{4, 12, 20\}$

Values for $\mathbf{p}_c$ were chosen in dependence of n:

1. If $n = 100$: $\mathbf{p}_c \in \{0.01, 0.03, 0.05\}$
2. If $n = 500$: $\mathbf{p}_c \in \{0.005, 0.015, 0.025\}$
3. If $n = 1000$: $\mathbf{p}_c \in \{0.001, 0.003, 0.005\}$

The three chosen values for $\mathbf{p}_c$ imply for any string length that, on average, 1%, 3%, or 5% of the string positions are exchanged. For each combination of values for the three parameters n, m and $|\Sigma|$ we randomly generated 10 problem instances. This results in a total of 810 benchmark instances. To test each instance with different limits for the number of allowed bad columns, we used values for k from $\{2, n/20, n/10\}$.

5.2 Tuning of LNS

The automatic configuration tool irace [6] was used to tune LNS concerning several parameters for which well-working values had to be found: (1) the lower

Table 1: Parameters settings produced by irace for Lns concerning $|\Sigma| = 4$.

	k=2		**k=n/20**		**k=n/10**	
n	(l,u)	$\mathbf{t}_{\max}$	(l,u)	$\mathbf{t}_{\max}$	(l,u)	$\mathbf{t}_{\max}$
100	(70,70)	20	(90,90)	10	(90,90)	10
500	(90,90)	10	(90,90)	15	(90,90)	20
1000	(90,90)	5	(90,90)	15	(90,90)	10

Table 2: Parameters settings produced by irace for Lns concerning $|\Sigma| = 12$.

	k=2		**k=n/20**		**k=n/10**	
n	(l,u)	$\mathbf{t}_{\max}$	(l,u)	$\mathbf{t}_{\max}$	(l,u)	$\mathbf{t}_{\max}$
100	(80,80)	20	(90,90)	15	(40,40)	20
500	(90,90)	5	(90,90)	20	(90,90)	5
1000	(90,90)	15	(90,90)	20	(90,90)	20

Table 3: Parameters settings produced by irace for Lns concerning $|\Sigma| = 20$.

	k=2		**k=n/20**		**k=n/10**	
n	(l,u)	$\mathbf{t}_{\max}$	(l,u)	$\mathbf{t}_{\max}$	(l,u)	$\mathbf{t}_{\max}$
100	(90,90)	10	(90,90)	20	(70,70)	20
500	90,90]	15	(90,90)	10	(90,90)	5
1000	(90,90)	10	(90,90)	20	(90,90)	20

and upper bounds $\mathbf{perc}^{\mathrm{l}}_{\mathrm{dest}}$ and $\mathbf{perc}^{\mathrm{u}}_{\mathrm{dest}}$ of the percentage of strings to be deleted from the current incumbent solution S_{bsf} and (2) the maximum CPU-time $t_{\max}$ (in seconds) allowed for CPLEX per application within Lns. irace was applied separately for each combination of values for n, $|\Sigma|$ and k, respectively. Note that no separate tuning was performed concerning the string length m and the percentage of character change $\mathbf{p}_c$. This is because, after initial runs, it was shown that parameters n, $|\Sigma|$ and k have a bigger influence on the behavior of the algorithm than m and $\mathbf{p}_c$. Summarizing, irace was applied 27 times with a budget of 1000 applications of Lns per tuning run. For each application of Lns a time limit of $n/2$ CPU seconds was given. For each run of irace, one tuning instance for each combination of m and $\mathbf{p}_c$ was generated. This gives a total of 9 tuning instances per run of irace.

The parameter value ranges chosen for the tuning processes of the Lns are as follows:

- For the lower and upper bound values of the destruction percentage, the following value combinations were considered: $(\mathbf{perc}^{\mathrm{l}}_{\mathrm{dest}}, \mathbf{perc}^{\mathrm{u}}_{\mathrm{dest}}) \in$ {(10,10), (20,20), (30,30), (40,40), (50,50), (60,60), (70,70), (80,80), (90,90), (10,30), (10,50), (30,50), (30,70)}. Note that in those cases in which both bounds have the same value, the percentage of deleted nodes is always the same.
- $\mathbf{t}_{\max} \in \{5.0, 10.0, 15.0, 20.0\}$.

The results of the tuning processes are shown in Tables 1, 2, and 3. The following observations can be made. First, nearly in all cases setting $(90, 90)$ is chosen for the lower and upper bounds of the destruction percentage. This is, at first, surprising, because generally LNS algorithms require a lower setting of the destruction percentage. Later, at the end of Section 5.3 we will study why such a high setting for the destruction percentage is chosen in this case. Second, no clear trend can be extracted from the chosen settings for $t_{\max}$. The reason for this is related to the reason for choosing a high destruction percentage, which will be outlined later, as mentioned above. Finally, note that, in general, the setting of $t_{\max}$ is very much dependent on the specification of the machines used for performing the experimentation. Therefore, before running this algorithm on different machines, the tuning process must be repeated.

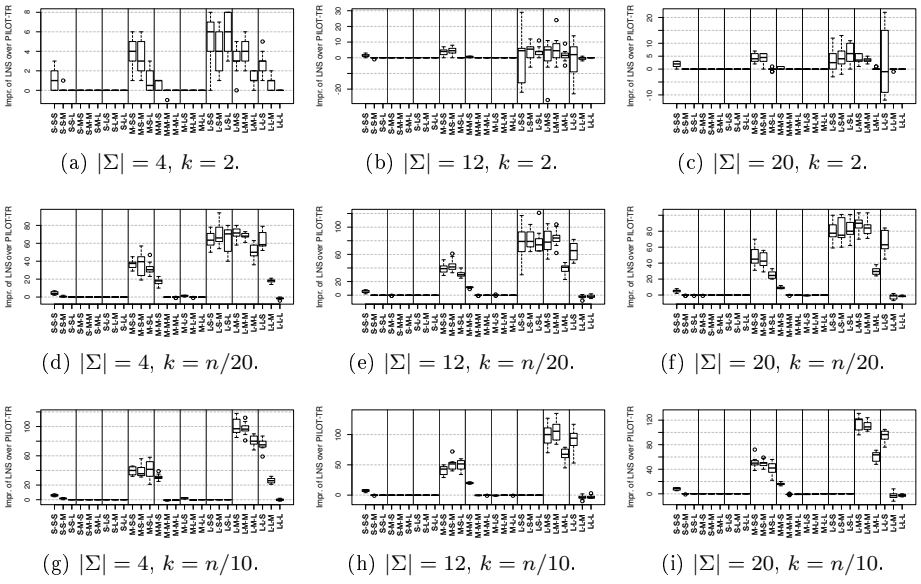
5.3 Results

The results are presented in numerical form in three tables: Table 4 contains the results for all instances with $|\Sigma| = 4$, Table 5 contains the results for all instances with $|\Sigma| = 12$, and Table 6 shows the results for instances with $|\Sigma| = 20$. All three tables have the following format. The first three table columns indicate the number of input strings (n), the string length (m) and the change probability ($\mathbf{p}_c$). The results of Greedy-F, Greedy-Ms, Lns and Cplex are presented in three groups of columns, one group for each of the three values for k. In each group of columns, the results are given in the following way. For Greedy-F, Greedy-Ms and Lns we provide the average of the best objective values obtained for the 10 random instances of each table row (columns with heading "mean"), and the corresponding average of the computation times in seconds (columns with heading "time"). Hereby, Lns was applied with a time limit of $n/2$ CPU seconds to each problem instance. For Cplex, which was applied with the same time limits as Lns to each problem instance, we provide the average objective values (column with heading "mean") and the corresponding average optimality gaps (column with heading "gap"). Note that in those cases in which this gap has value zero, the 10 corresponding problem instances were solved to optimality within the allowed computation time limit. Finally, note that the best result of each table row is shown in bold font.

Apart from the numerical results provided in the form of tables, the graphics of Figure 1 show the improvement of Lns over Greedy-Ms and the graphics of Figure 2 show the improvement of Lns over Cplex. The notation X-Y-Z on the x-axis of these graphics has the following meaning. X, Y, and Z take values from $\{\mathrm{S}, \mathrm{M}, \mathrm{L}\}$, where S refers to *small*, M refers to *medium* and L refers to *large*. While X refers to the number of input strings, Y refers to their length, and Z to the probability of change. In case of positions X and Y, S refers to 100, M to 500, and L to 1000, while in the case of Z, S refers to $\{0.01, 0.005, 0.001\}$, M to $\{0.03, 0.015, 0.003\}$, and L to $\{0.05, 0.025, 0.005\}$, respectively, depending on the value of n.

The experimental results allow us to make the following observations:

- First, no substantial differences can be observed concerning the relative performance of the algorithms for what concerns instances of different alphabet sizes. The only difference is that the objective function values



(a) $|\Sigma| = 4$, $k = 2$. (b) $|\Sigma| = 12$, $k = 2$. (c) $|\Sigma| = 20$, $k = 2$.

(d) $|\Sigma| = 4$, $k = n/20$. (e) $|\Sigma| = 12$, $k = n/20$. (f) $|\Sigma| = 20$, $k = n/20$.

(g) $|\Sigma| = 4$, $k = n/10$. (h) $|\Sigma| = 12$, $k = n/10$. (i) $|\Sigma| = 20$, $k = n/10$.

Figure 1: Improvement of Lns over Greedy-Ms (in absolute terms). Each box shows these differences for the corresponding 10 instances. Note that negative values indicate that Greedy-Ms obtained a better result than Lns.

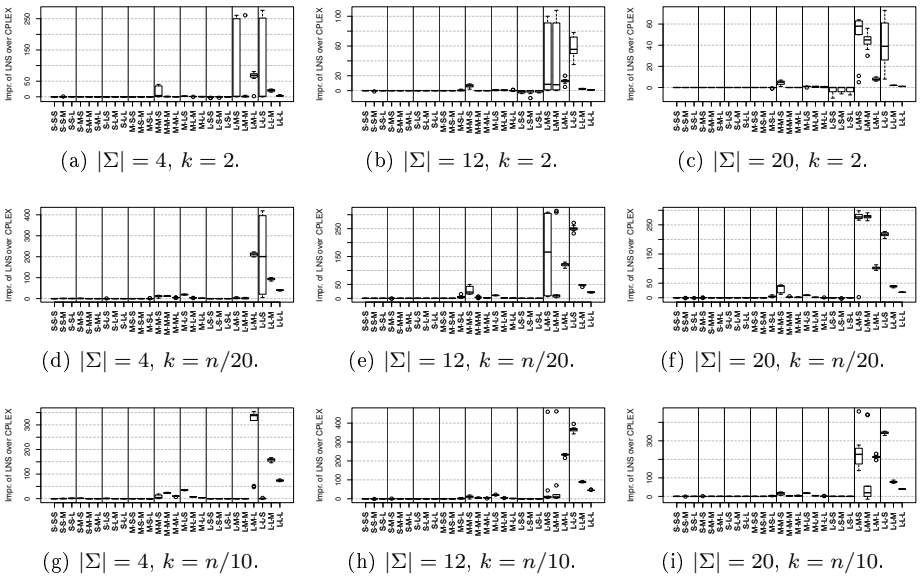
generally decrease with increasing alphabet size.

- Greedy-Ms and Lns both clearly outperform Greedy-F w.r.t. solution quality. In particular, it clearly pays of to start the greedy strategy from n different partial solutions (each one containing exactly one of the n input strings) instead of applying the greedy strategy only to the partial solution containing the string obtained by the frequency-based mechanism.

- We can also observe that Lns and Greedy-Ms perform comparably for the small problem instances w.r.t. solution quality. However, concerning large problem instances—especially in the case of $|\Sigma| = 4$—Lns generally outperforms Greedy-Ms. Nevertheless, there are some noticeable exceptions, especially for what concerns cases L-S-S and L-L-S in Figure 1b and case L-L-S in Figure 1c. That is, when $k = 2$ and problem instances are large, Greedy-Ms sometimes performs better then Lns.

- Concerning Lns in comparison to Cplex, we can see the relative behavior that is to be expected nicely displayed in the graphics of Figure 2. In particular, Lns is competitive with Cplex for small and medium size instances of all alphabet sizes and for all values of k. Moreover, Lns generally outperforms Cplex in the context of larger problem instances. This is, again, with the exception of instances L-S-$*$ both in the case of $|\Sigma| \in \{12, 20\}$ for $k = 2$ where Cplex performs slightly better than Lns.

- Concerning computation time requirements, Cplex requires substantially more computation time than Lns, Greedy-R and Greedy-Ms for reaching solutions of similar quality. However, keep in mind that this statement is bound to the machines used for the experimentation.



(a) $|\Sigma| = 4$, $k = 2$. (b) $|\Sigma| = 12$, $k = 2$. (c) $|\Sigma| = 20$, $k = 2$.

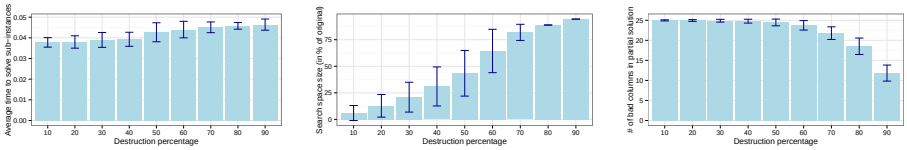
(d) $|\Sigma| = 4$, $k = n/20$. (e) $|\Sigma| = 12$, $k = n/20$. (f) $|\Sigma| = 20$, $k = n/20$.

(g) $|\Sigma| = 4$, $k = n/10$. (h) $|\Sigma| = 12$, $k = n/10$. (i) $|\Sigma| = 20$, $k = n/10$.

Figure 2: Improvement of LNS over CPLEX (in absolute terms). Each box shows these differences for the corresponding 10 instances. Note that negative values indicate that CPLEX obtained a better result than LNS.

Summarizing, we can say that, even though LNS generally outperforms the other considered techniques, the disadvantages of LNS in comparison to GREEDY-MS and CPLEX for instances with $|\Sigma| \in \{12, 20\}$ and $k = 2$ leave certainly room for improvement.

Finally, we also want to study the reasons for the fact that the tuning procedure (irace) has chosen a very high percentage of destruction in nearly all cases. Remember that this is rather unusual for an LNS algorithm. In order to shed some light on this matter, we applied LNS for 100 iterations to two exemplary cases: (1) a problem instance with $n = 500$, $m = 100$, $\mathbf{p}_c = 0.015$, $|\Sigma| = 4$, $k = n/20$, and (2) a problem instance with $n = 1000$, $m = 500$, $\mathbf{p}_c = 0.003$, $|\Sigma| = 4$, $k = n/20$. This was done in both cases for fixed destruction percentages over the whole range between 10% and 90%. In all runs we took the following three different measures: (1) the average time (in seconds) needed by CPLEX for obtaining the optimal solutions when applied to the partial solution of each iteration within LNS, (2) the average size (in percent of the size of the original problem instance) of the considered sub-instances, and (3) the average number of bad columns that are already determined by the fixed strings in the sub-instances. These three measures are displayed for each considered destruction percentage in three different graphics of Figure 3 (concerning the first example instance) and Figure 4 (concerning the second example instance). It can be observed that, in both cases, CPLEX is very fast for any destruction percentage. In fact, the time difference between destruction percentages 10% and 90% is nearly negligible. Moreover, note that CPLEX requires 18.1 seconds for solving the original problem instance of the first case to optimality, whereas in the second case CPLEX is not able to find a solution with an opti-



(a) (b) (c)

Figure 3: Concerning the whole range of considered destruction percentages, the three graphics show (a) the average time (in seconds) used by CPLEX within Lns for each application at each iteration, (b) the average size (in percent of the size of the original problem instance) of the considered sub-instances, and (c) the average number of bad columns that are already determined by the fixed strings in the sub-instances. All graphics concern an example instance with $n = 500$, $m = 100$, $\mathbf{p}_c = 0.015$, $|\Sigma| = 4$, $k = n/20$.

mality gap below 100 within 250 CPU seconds. This implies that in the case of the MSFBC problem, as long as a small partial solution is given, CPLEX is very fast in deriving the optimal solution that contains the respective partial solution. Surprisingly, there is hardly any difference in the computation time requirements with respect to the size of this partial solution, even though the size of the sub-instances varies considerably for different destruction percentages (see Figures 3b and 4b). Finally, observe that the number of bad columns that are already determined by the strings contained in a sub-instance generated with a rather low destruction percentage is very close to the maximum number of allowed bad columns (25 in the case of the first example instance, and 50 in the case of the second example instance). This can be observed in Figures 3c and 4c. In other words, the chance to generate a better solution on the basis of low destruction percentage is rather small.

Summarizing, these observations explain why irace chose a large destruction percentage nearly in all cases. This also implies that the computation time limit given to CPLEX within Lns does hardly play any role. In other words, all computation time limits that we considered where generally sufficient for CPLEX when applied within Lns. This is why no tendency can be observed in the values chosen by irace for $t_{\max}$.

6 Conclusions and Outlook to Future Work

In this paper we considered the most strings with few bad columns problem, which is an NP-hard combinatorial optimization problem from the bioinformatics field. A hybrid metaheuristics approach was proposed that makes use of a new integer linear programming model within a large neighborhood search algorithm. After comparing the large neighborhood search algorithm with two greedy strategies and the stand-alone application of CPLEX, we can say that the large neighborhood search algorithm is, apart from a few exceptional cases, a new state-of-the-art algorithm for the considered problem.

Concerning future work we plan to consider other ways of combining metaheuristic strategies with CPLEX. One option concerns, for example, the use of CPLEX for deriving a complete solution on the basis of a partial solution within

Table 4: Experimental results for instances with $|\Sigma| = 4$ (all times in seconds).

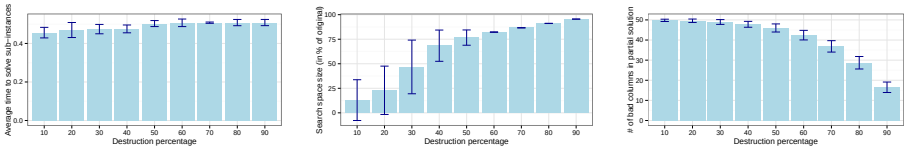
			$k = 2$								$k = n/20$								$k = n/10$							
			Greedy-F		Greedy-Ms		Lns		Cplex		Greedy-F		Greedy-Ms		Lns		Cplex		Greedy-F		Greedy-Ms		Lns		Cplex	
n	m	$\mathbf{p}_c$	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**
		0.01	29.0	0.0	31.0	0.1	**32.1**	0.1	**32.1**	0.0	33.5	0.0	35.3	0.0	**39.6**	0.1	**39.6**	0.0	42.2	0.0	43.8	0.0	**49.7**	1.7	**49.7**	0.0
	100	0.03	2.5	0.0	3.9	0.2	**4.0**	0.1	3.9	>99.9	4.4	0.0	7.6	0.1	**8.2**	0.1	7.5	>99.9	7.7	0.0	13.2	0.2	**14.8**	0.2	14.3	99.1
		0.05	1.9	0.0	**2.1**	0.0	**2.1**	0.0	1.6	>99.9	2.6	0.0	**4.2**	0.0	**4.2**	0.1	3.7	>99.9	4.2	0.0	**8.0**	0.1	**8.0**	0.2	6.9	>99.9
		0.01	**1.4**	0.0	**1.4**	0.1	**1.4**	0.0	1.0	>99.9	2.2	0.0	**3.2**	0.1	**3.2**	0.1	2.4	>99.9	3.6	0.0	**5.8**	0.4	**5.8**	0.1	4.5	>99.9
100	500	0.03	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
		0.05	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.1	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
		0.01	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.1**	0.0	**1.1**	0.2	**1.1**	1.0	1.0	>99.9	2.0	0.0	**2.1**	0.2	**2.1**	0.0	1.3	>99.9
	1000	0.03	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9
		0.05	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9
		0.005	139.3	0.0	143.0	2.0	**147.0**	1.7	**147.0**	0.0	244.5	0.0	248.2	16.4	**284.9**	5.2	**284.9**	0.0	346.7	0.0	350.6	20.4	**390.0**	2.7	**390.0**	0.0
	100	0.015	137.2	0.0	141.7	2.0	**145.2**	2.1	**145.2**	0.0	242.8	0.0	247.5	12.9	**284.2**	4.7	**248.2**	0.0	348.5	0.0	351.8	20.4	**390.8**	2.3	**390.8**	0.0
		0.025	39.4	0.0	44.9	1.8	**45.8**	1.5	**45.8**	77.4	118.5	0.0	129.1	16.6	**160.7**	21.6	160.3	29.6	228.4	0.0	243.7	28.7	285.6	19.2	**286.1**	4.9
		0.005	33.3	0.0	36.6	7.1	**37.3**	3.2	21.4	>99.9	56.8	0.0	69.7	79.5	**86.9**	13.9	75.8	89.9	84.5	0.0	104.0	114.0	**135.1**	14.4	127.2	38.8
500	500	0.015	1.2	0.0	**1.3**	2.0	1.2	0.0	0.5	>99.9	6.3	0.0	**12.3**	20.7	**12.3**	8.7	0.3	>99.9	12.0	0.0	**24.2**	54.4	23.4	30.1	0.0	>99.9
		0.025	**1.0**	0.0	**1.0**	2.0	**1.0**	0.0	0.7	>99.9	3.7	0.0	**5.4**	8.6	5.2	0.0	0.4	>99.9	6.7	0.0	**11.4**	19.3	10.9	20.1	0.9	>99.9
		0.005	2.2	0.0	**2.3**	4.0	**2.3**	0.0	0.4	>99.9	8.8	0.0	18.2	62.2	**19.5**	13.8	0.3	>99.9	16.1	0.0	32.6	113.8	**34.7**	13.9	0.1	>99.9
	1000	0.015	**1.0**	0.1	**1.0**	3.9	**1.0**	0.0	0.1	>99.9	2.8	0.0	**3.9**	11.5	3.8	0.0	0.0	>99.9	5.1	0.0	**7.9**	26.6	7.6	5.1	0.1	>99.9
		0.025	**1.0**	0.1	**1.0**	3.9	**1.0**	0.0	0.4	>99.9	**2.0**	0.1	**2.0**	4.1	**2.0**	0.0	0.3	>99.9	3.0	0.1	**4.0**	12.3	**4.0**	0.0	0.0	>99.9
		0.001	270.4	0.0	276.9	10.6	281.9	15.9	**282.6**	0.0	664.3	0.0	668.8	87.1	**733.0**	42.5	**733.0**	0.0	**1000**	0.0	**1000.0**	106.5	**1000.0**	0.0	**1000.0**	0.0
	100	0.003	271.4	0.0	276.1	8.3	280.2	48.9	**280.5**	0.0	660.6	0.0	666.1	86.6	**735.9**	32.5	**735.9**	0.0	**1000**	0.0	**1000.0**	106.4	**1000.0**	0.0	**1000.0**	0.0
		0.005	265.8	0.0	274.4	8.3	**280.4**	32.6	**280.4**	0.0	666.0	0.0	671.9	87.4	**737.1**	27.3	**737.1**	0.0	**1000**	0.0	**1000.0**	106.2	**1000.0**	0.0	**1000.0**	0.0
		0.001	257.7	0.1	260.9	34.3	**264.1**	13.5	29.8	>99.9	372.0	0.1	375.8	463.0	**446.9**	22.2	397.8	>99.9	487.7	0.1	491.0	835.5	**590.6**	16.2	529.3	>99.9
1000	500	0.003	254.0	0.1	258.2	34.2	**262.3**	14.5	24.3	>99.9	372.0	0.1	375.9	470.7	**444.5**	35.7	399.3	>99.9	485.6	0.1	488.9	839.4	**586.1**	18.8	**586.1**	0.0
		0.005	63.6	0.1	68.7	30.1	**69.8**	5.8	6.6	>99.9	134.5	0.1	160.7	544.0	**211.8**	54.0	0.3	>99.9	221.7	0.0	257.2	1022.5	**338.0**	60.5	0.0	>99.9
		0.001	253.0	0.1	256.3	61.4	**259.1**	17.1	127.9	>99.9	334.0	0.1	337.8	836.3	**400.6**	25.4	111.8	>99.9	422.3	0.1	425.4	1549.1.2	**500.0**	23.9	0.0	>99.9
	1000	0.003	17.7	0.1	19.9	36.2	**20.6**	10.2	0.0	>99.9	45.5	0.1	75.6	833.0	**93.5**	51.6	0.0	>99.9	78.7	0.1	131.8	1586.8	**158.2**	60.2	0.0	>99.9
		0.005	3.0	0.1	**3.6**	17.3	**3.6**	0.9	0.0	>99.9	16.3	0.1	**42.2**	578.3	40.6	24.2	0.0	>99.9	32.8	0.1	74.0	1025.7	**74.1**	91.9	0.0	>99.9

Table 5: Experimental results for instances with $|\Sigma| = 12$ (all times in seconds).

			$k=2$								$k=n/20$								$k=n/10$							
			Greedy-F		Greedy-Ms		Lns		Cplex		Greedy-F		Greedy-Ms		Lns		Cplex		Greedy-F		Greedy-Ms		Lns		Cplex	
n	m	$\mathbf{p}_c$	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**
100	100	0.01	13.3	0.0	15.4	0.1	**16.8**	0.1	**16.8**	0.0	18.6	0.0	21.0	0.1	**25.9**	0.1	**25.9**	0.0	27.9	0.0	29.9	0.2	**37.3**	0.1	**37.3**	0.0
		0.03	1.6	0.0	**1.9**	0.0	1.8	5.0	**1.9**	0.0	2.7	0.0	**4.4**	0.1	**4.4**	0.1	4.3	0.0	4.5	0.9	**8.8**	0.1	8.7	0.3	**8.8**	0.0
		0.05	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	1.9	0.0	**2.0**	0.0	**2.0**	5.0	**2.0**	87.3	3.0	0.0	**4.0**	0.1	**4.0**	0.9	**4.0**	0.0
	500	0.01	**1.0**	0.0	**1.0**	1.0	**1.0**	0.0	**1.0**	0.0	1.5	0.0	**1.7**	0.5	1.5	0.0	**1.7**	>99.9	2.7	0.0	**3.2**	0.2	**3.2**	0.0	3.1	>99.9
		0.03	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
		0.05	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
	1000	0.01	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.1**	0.0	**1.1**	0.2	**1.1**	0.0	**1.1**	>99.9
		0.03	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0
		0.05	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0
500	100	0.005	51.2	0.0	56.6	2.1	**60.5**	6.7	**60.5**	0.0	180.2	0.0	184.9	15.8	**224.4**	7.2	**224.4**	0.0	307.8	0.0	310.9	25.8	**352.1**	3.4	**352.1**	0.0
		0.015	50.2	0.0	54.9	2.1	**59.1**	3.1	**59.1**	0.0	173.7	0.0	178.6	16.0	**222.5**	5.8	**222.5**	0.0	298.1	0.0	302.0	26.0	**353.1**	3.1	**353.1**	0.0
		0.025	7.4	0.0	**11.4**	1.3	**11.4**	6.2	10.9	>99.9	66.6	0.0	73.6	16.3	**104.4**	23.0	99.6	85.6	168.9	0.0	183.9	30.8	**233.2**	14.6	231.4	22.4
	500	0.005	5.8	0.0	7.9	6.0	**8.4**	2.2	2.0	>99.9	23.1	0.0	35.2	54.2	**46.6**	10.9	17.3	>99.9	44.8	0.0	64.6	102.2	**84.3**	34.8	72.7	>99.9
		0.015	**1.0**	0.0	**1.0**	2.6	**1.0**	0.0	**1.0**	>99.9	4.9	0.0	**7.2**	12.2	7.0	0.0	3.6	>99.9	8.9	0.0	**15.0**	26.6	14.6	10.1	10.1	>99.9
		0.025	**1.0**	0.0	**1.0**	2.0	**1.0**	0.0	**1.0**	>99.9	2.8	0.0	**3.0**	5.5	**3.0**	0.0	1.2	>99.9	5.3	0.0	**7.0**	12.7	6.9	0.0	1.6	>99.9
	1000	0.005	**1.0**	0.0	**1.0**	3.9	**1.0**	0.0	0.3	>99.9	6.3	0.1	**10.5**	35.7	**10.5**	13.0	0.0	>99.9	11.9	0.1	**20.3**	68.6	19.6	26.0	0.0	>99.9
		0.015	**1.0**	0.1	**1.0**	3.9	**1.0**	0.0	0.5	>99.9	2.0	0.1	**2.1**	7.7	**2.1**	0.0	0.3	>99.9	4.0	0.1	**5.0**	16.9	**5.0**	0.0	0.1	>99.9
		0.025	**1.0**	0.1	**1.0**	4.0	**1.0**	0.0	0.8	>99.9	**1.0**	0.1	**1.0**	4.0	**1.0**	0.0	0.5	>99.9	2.0	0.1	**2.2**	7.8	2.1	0.0	**2.2**	>99.9
1000	100	0.001	105.5	0.0	110.8	8.5	112.1	24.0	**116.2**	18.6	575.8	0.0	582.7	105.9	**661.0**	57.4	658.5	0.0	**1000.0**	0.0	**1000.0**	163.0	**1000.0**	0.1	**1000.0**	0.0
		0.003	100.2	0.0	107.4	8.5	111.7	26.2	**113.4**	19.1	572.9	0.0	578.9	105.9	**660.5**	47.4	**660.5**	0.0	**1000.0**	0.0	**1000.0**	130.4	**1000.0**	0.1	**1000.0**	0.0
		0.005	105.4	0.0	110.5	8.5	114.4	45.7	**115.2**	19.1	576.8	0.0	582.3	105.4	**660.8**	48.1	**660.8**	0.0	**1000.0**	0.0	**1000.0**	129.9	**1000.0**	0.1	**1000.0**	0.0
	500	0.001	88.3	0.1	93.9	37.0	**95.0**	13.6	72.2	>99.9	222.0	0.1	226.9	600.6	**305.3**	58.5	0.0	>99.9	355.7	0.1	360.5	1086.1	460.5	34.6	**460.8**	10.5
		0.003	91.3	0.1	95.4	36.9	**100.8**	17.3	55.8	>99.9	223.5	0.1	227.0	590.7	**311.4**	62.1	0.0	>99.9	358.1	0.1	361.9	1073.2	**469.6**	37.4	320.9	>99.9
		0.005	9.0	0.1	12.2	24.3	**13.6**	7.9	0.0	>99.9	59.0	0.1	82.1	492.0	**120.9**	65.4	0.0	>99.9	128.5	0.1	165.8	981.1	**232.4**	81.2	0.0	>99.9
	1000	0.001	91.6	0.2	**95.0**	70.5	94.3	29.6	34.8	>99.9	181.6	0.2	185.4	1382.6	**250.7**	65.7	0.0	>99.9	272.5	0.0	275.7	1966.1	**366.0**	27.3	0.0	>99.9
		0.003	2.1	0.2	**2.7**	17.6	2.4	0.7	0.0	>99.9	21.4	0.2	**49.2**	673.0	46.7	32.1	0.0	>99.9	44.4	0.2	**93.1**	1292.1	89.5	47.8	0.0	>99.9
		0.005	**1.0**	0.2	**1.0**	15.3	**1.0**	0.0	0.0	>99.9	12.4	0.2	**23.7**	323.6	21.7	10.4	0.0	>99.9	24.2	0.2	**47.8**	663.4	44.9	5.3	0.0	>99.9

Table 6: Experimental results for instances with $|\Sigma| = 20$ (all times in seconds).

			$k=2$								$k=n/20$								$k=n/10$							
			Greedy-F		Greedy-Ms		Lns		Cplex		Greedy-F		Greedy-Ms		Lns		Cplex		Greedy-F		Greedy-Ms		Lns		Cplex	
n	m	$\mathbf{p}_c$	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**time**	**mean**	**gap**
100	100	0.01	8.7	0.0	10.5	0.1	**12.2**	0.1	**12.2**	0.0	14.6	0.0	16.4	0.1	**21.5**	0.2	**21.5**	0.0	23.8	0.0	26.0	0.3	**34.0**	0.1	**34.0**	0.0
		0.03	**1.5**	0.0	**1.5**	0.0	**1.5**	0.0	**1.5**	0.0	2.9	0.0	3.8	0.0	3.8	0.1	**3.9**	0.0	4.9	0.0	**8.4**	0.1	8.3	0.2	**8.4**	0.0
		0.05	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	1.8	0.0	**1.9**	0.0	1.8	0.0	**1.9**	0.0	3.0	0.0	**3.9**	0.1	**3.9**	0.8	**3.9**	>99.9
	500	0.01	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	1.2	0.0	**1.3**	0.1	1.2	0.0	1.2	>99.9	2.5	0.0	**3.0**	0.2	**3.0**	0.0	2.9	>99.9
		0.03	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
		0.05	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
	1000	0.01	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.1	**1.0**	0.0	**1.0**	>99.9
		0.03	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9
		0.05	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	0.0	**1.0**	0.0	**1.0**	0.2	**1.0**	0.0	**1.0**	>99.9
500	100	0.005	35.3	0.0	39.7	2.0	**44.1**	9.3	**44.1**	0.0	161.9	0.0	166.1	16.3	**214.8**	9.8	**214.8**	0.0	295.4	0.0	299.1	26.7	**350.1**	4.3	**350.1**	0.0
		0.015	34.7	0.0	40.1	2.0	**44.0**	7.4	**44.0**	0.0	164.1	0.0	170.3	16.2	**213.6**	6.7	**213.6**	0.0	294.5	0.0	298.0	26.5	**347.5**	4.5	**347.5**	0.0
		0.025	3.4	0.0	7.4	1.1	7.3	2.2	**7.5**	>99.9	58.2	0.0	71.7	15.8	**96.9**	30.2	92.5	94.7	168.9	0.0	183.4	30.4	**225.0**	29.1	223.1	24.9
	500	0.005	3.2	0.1	5.3	5.0	**5.7**	3.8	1.2	>99.9	17.9	0.1	31.7	52.0	**41.0**	11.1	8.4	>99.9	37.1	0.1	61.5	99.5	**77.7**	21.3	61.7	>99.9
		0.015	**1.0**	0.1	**1.0**	2.0	**1.0**	0.0	**1.0**	>99.9	4.1	0.1	**6.1**	10.2	6.0	0.0	4.0	>99.9	8.5	0.1	**13.8**	23.5	12.8	5.0	10.0	>99.9
		0.025	**1.0**	0.1	**1.0**	2.0	**1.0**	0.0	**1.0**	>99.9	2.3	0.1	**3.0**	4.5	**3.0**	0.0	1.6	>99.9	5.0	0.1	**6.8**	11.6	6.2	0.0	3.3	>99.9
	1000	0.005	**1.0**	0.1	**1.0**	3.8	**1.0**	0.0	0.1	>99.9	5.9	0.1	**9.0**	29.5	8.5	13.6	0.0	>99.9	11.1	0.1	**18.1**	61.5	17.4	15.9	0.0	>99.9
		0.015	**1.0**	0.1	**1.0**	3.9	**1.0**	0.0	0.4	>99.9	**2.0**	0.1	**2.0**	7.6	**2.0**	0.0	0.6	>99.9	4.0	0.1	**4.5**	15.4	4.2	0.0	0.4	>99.9
		0.025	**1.0**	0.1	**1.0**	5.1	**1.0**	0.0	0.7	>99.9	**1.0**	0.1	**1.0**	4.0	**1.0**	0.0	0.9	>99.9	**2.0**	0.1	**2.0**	7.8	**2.0**	0.0	0.8	>99.9
1000	100	0.001	71.3	0.0	77.6	8.4	80.6	29.6	**82.2**	24.4	560.0	0.0	565.8	109.2	**645.0**	52.0	**645.0**	0.0	**1000.0**	0.0	**1000.0**	134.8	**1000.0**	0.1	**1000.0**	0.0
		0.003	68.1	0.0	75.3	8.4	79.9	35.6	**80.9**	76.5	558.3	0.0	565.7	109.7	646.2	60.4	**646.7**	0.0	**1000.0**	0.0	**1000.0**	135.3	**1000.0**	0.1	**1000.0**	0.0
		0.005	66.9	0.0	76.1	8.4	82.0	39.3	**83.6**	21.4	561.4	0.0	566.2	110.6	**648.2**	61.4	**648.2**	0.0	**1000.0**	0.0	**1000.0**	135.0	**1000.0**	0.1	**1000.0**	0.0
	500	0.001	55.7	0.1	60.7	37.1	**64.6**	19.0	16.0	>99.9	188.2	0.1	193.3	653.3	**281.7**	55.1	29.4	>99.9	323.9	0.1	328.3	1127.6	**443.6**	31.2	208.5	>99.9
		0.003	53.4	0.1	56.4	37.1	**59.9**	19.7	15.5	>99.9	189.9	0.1	193.6	618.9	**277.9**	34.9	0.0	>99.9	324.7	0.1	328.9	1170.7	**439.8**	31.5	339.3	>99.9
		0.005	4.7	0.1	7.7	22.3	**7.9**	18.7	0.0	>99.9	47.7	0.1	72.7	476.4	**102.5**	87.9	0.0	>99.9	111.1	0.1	151.7	931.3	**212.8**	70.4	0.0	>99.9
	1000	0.001	54.1	0.2	58.0	69.8	**59.8**	33.6	17.8	>99.9	147.2	0.2	150.7	1111.7	**217.0**	61.1	0.0	>99.9	244.6	0.2	248.0	2840.9	**342.0**	22.2	0.0	>99.9
		0.003	2.0	0.2	**2.1**	16.1	2.0	0.0	0.0	>99.9	19.6	0.2	**41.2**	569.3	39.1	93.5	0.0	>99.9	40.3	0.2	**81.6**	1099.8	78.3	79.8	0.0	>99.9
		0.005	**1.0**	0.2	**1.0**	15.3	**1.0**	0.0	0.0	>99.9	11.9	0.2	**20.2**	275.1	18.8	0.1	0.0	>99.9	23.7	0.2	**42.7**	582.7	39.9	16.1	0.0	>99.9



(a) (b) (c)

Figure 4: Concerning the whole range of considered destruction percentages, the three graphics show (a) the average time (in seconds) used by CPLEX within Lns for each application at each iteration, (b) the average size (in percent of the size of the original problem instance) of the considered sub-instances, and (c) the average number of bad columns that are already determined by the fixed strings in the sub-instances. All graphics concern an example instance with $n = 1000$, $m = 500$, $\mathbf{p}_c = 0.003$, $|\Sigma| = 4$, $k = n/20$.

the recombination operator of an evolutionary algorithm.

7 Acknowledgements

All experiments were executed in the High Performance Cluster managed by the Research and Development Lab (RDlab) of the Computer Science Dept. at the Universitat Politécnica de Catalunya (`http://rdlab.cs.upc.edu`). We thank all the RDlab staff for their support.

References

[1] Boucher, C., Landau, G.M., Levy, A., Pritchard, D., Weimann, O.: On approximating string selection problems with outliers. Theoretical Computer Science **498**, 107–114 (2013)

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

[3] Hsu, W.J., Du, M.W.: Computing a longest common subsequence for a set of strings. BIT Numerical Mathematics **24**(1), 45–59 (1984). DOI 10.1007/BF01934514

[4] Landau, G.M., Schmidt, J.P., Sokol, D.: An algorithm for approxixmate tandem repeat. Journal of Computational Biology **8**(1), 1–18 (2001)

[5] Lizarraga, E., Blesa, M.J., Blum, C., Raidl, G.R.: On solving the most strings with few bad columns problem: An ILP model and heuristics. In: Proceedings of INISTA 2015 – International Symposium on Innovations in Intelligent SysTems and Applications, pp. 1–8. IEEE Press (2015)

[6] 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)

[7] 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)

[8] Mousavi, S., Babaie, M., Montazerian, M.: An improved heuristic for the far from most strings problem. Journal of Heuristics **18**, 239–262 (2012)

[9] Pappalardo, E., Pardalos, P.M., Stracquadanio, G.: Optimization approaches for solving string selection problems. SpringerBriefs in Optimization. Springer New York (2013)

[10] Pisinger, D., Ropke, S.: Large neighborhood search. In: M. Gendreau, J.Y. Potvin (eds.) Handbook of Metaheuristics, *International Series in Operations Research & Management Science*, vol. 146, pp. 399–419. Springer US (2010)

[11] Rajasekaran, S., Hu, Y., Luo, J., Nick, H., Pardalos, P.M., Sahni, S., Shaw, G.: Efficient algorithms for similarity search. Journal of Combinatorial Optimization **5**(1), 125–132 (2001)

[12] Rajasekaran, S., Nick, H., Pardalos, P.M., Sahni, S., Shaw, G.: Efficient algorithms for local alignment search. Journal of Combinatorial Optimization **5**(1), 117–124 (2001)

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

[14] Voß, S., Fink, A., Duin, C.: Looking ahead with the pilot method. Annals of Operations Research **136**(1), 285–302 (2005)