On Solving The Most Strings With Few Bad Columns Problem: An ILP Model and Heuristics

Evelia Lizárraga and Maria J. Blesa
Dept. of Computer Science
Technical Univ. of Catalonia (UPC)
Barcelona, Spain
Email: {evelial,mjblesa}@cs.upc.edu

Christian Blum
Dept. of Computer Science and
Artificial Intelligence
Univ. of the Basque Country UPV/EHU
and IKERBASQUE
San Sebastian, Spain
Email: christian.blum@ehu.es

Günther R. Raidl
Inst. of Computer Graphics and Algorithms
TU Wien, Vienna, Austria
Email: raidl@ads.tuwien.ac.at

***Abstract*—The most strings with few bad columns problem is an NP-hard combinatorial optimization problem from the bioinformatics field. This paper presents the first integer linear programming model for this problem. Moreover, a simple greedy heuristic and a more sophisticated extension, namely a greedy-based pilot method, are proposed. Experiments show that, as expected, the greedy-based pilot method improves over the greedy strategy. For problem instances of small and medium size the best results were obtained by solving the integer linear programming model by CPLEX, while the greedy-based pilot methods scales much better to large problem instances.**

I. Introduction

Optimization problems related to strings—such as protein or DNA sequences—are very common in bioinformatics. Examples include string selection problems [1]–[3], the longest common subsequence problem and its variants [4], [5], alignment problems [6], [7], and similarity search [8]. In this work we consider the so-called *most strings with few bad columns* (MSFBC) problem, which is an NP-hard combinatorial optimization problem. The problem was introduced in [9] in order to model the following situtation. Suppose that we are given a set of, for example, 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 MSFBC problem deals with separating the two subsets.

The 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 is henceforth denoted by $s_i[j]$. Moreover, given is a fixed value $k < m$. We are looking for a subset $S \subseteq I$ of maximal size such that the strings in S differ in at most k positions. Note that the strings of a subset $I \subseteq S$ are said to differ in a position $1 \leq j \leq m$ if, and only if, two strings $s_i, s_r \in S$ exist such that $s_i[j] \neq s_r[j]$. Finally, a position j in which the strings from S differ is called a *bad column*.[1]

The authors of [9] 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 [10]. As far as we know, no practical algorithm for solving the MSFBC problem has yet been proposed. The contribution of this paper is, first, an integer linear programming (ILP) model for the MSFBC problem. Second, we also propose a simple greedy heuristic as well as a greedy-based pilot method, which is an extension of the simple greedy heuristic. An extensive experimental evaluation shows that solving the ILP model with CPLEX is very competitive for instances of small and medium size. In contrast, large instances are better dealt with the greedy-based pilot method.

The remainder of this paper is organized as follows. Section II introduces the ILP model for the MSFBC problem, while Section III describes the proposed heuristics. An the experimental comparison is performed in Section IV, and Section V is dedicated to conclusions and an outlook to future work.

II. An ILP Model for the MSFBC Problem

For the description of the ILP model let $\Sigma_j := \{s_i[j] \mid i = 1, \ldots, n\}$, that is, $\Sigma_j \subseteq \Sigma$ is the set of letters appearing at least once at the j-th position of the n input strings. The ILP model for the MSFBC problem that we suggest makes use of several types of binary variables. First, the set of variables contains a binary variable x_i for each input string s_i. In case $x_i = 1$, the corresponding input string s_i is part of the solution, otherwise not. Furthermore, for each combination of a position j ($j = 1, \ldots, m$) and a letter $a \in \Sigma_j$ we use a binary variable z_j^a. Variable z_j^a is forced 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 postion $j = 1, \ldots, m$. 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.

[1]Note, in this context, that the set of input strings can be seen in form of a matrix in which the strings are the rows.

$$
\begin{array}{llll}
\max & \sum_{i=1}^{n} x_i & & (1) \\
\text{s.t.} & x_i \leq z_j^{s_i[j]} & \text{for } i = 1, \ldots, n & (2) \\
 & & \text{and } j = 1, \ldots, m & \\
 & \sum_{a \in \Sigma_j} z_j^a \leq 1 + |\Sigma_j| \cdot y_j & \text{for } j = 1, \ldots, m & (3) \\
 & \sum_{j=1}^{m} y_j \leq k & & (4) \\
 & x_i \in \{0, 1\} & \text{for } i = 1, \ldots, n & \\
 & z_j^a \in \{0, 1\} & \text{for } j = 1, \ldots, m & \\
 & & \text{and } a \in \Sigma_j & \\
 & y_j \in \{0, 1\} & \text{for } j = 1, \ldots, m &
\end{array}
$$

The objective function (1) maximizes the number of chosen strings. Constraints (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, constraints (3) ensure that y_j is set to one in case the selected strings differ at position j. Finally, constraint (4) ensures that not more than k bad columns are permitted.

III. Heuristic Approaches

In addition to applying the ILP solver CPLEX to the ILP model outlined in the previous section, we also developed two heuristic approaches: (1) a greedy algorithm, and (2) a greedy-based pilot method, which is an extension of the greedy algorithm with lookahead features.

A. Greedy Approach

The proposed greedy algorithm takes a partial solution S^p, which is a subset of the set I of input strings, as input. In the standalone version of the greedy algorithm this partial solution is empty. When used within the greedy-based pilot method outlined in Section III-B, the partial solution may be of any size. Henceforth, given a partial solution S^p, $\text{bc}(S^p)$ 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 fullfills the following two conditions:

1) $\text{bc}(S^p) \leq k$
2) There exist 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 fullfills the first one of these conditions.

The greedy algorithm, which is pseudo-coded in Algorithm 1, starts with a given partial solution S^p. At each iteration, exactly one of the strings from $I \setminus S^p$ is chosen, according to a greedy function, and added to S^p. The greedy function that is used concerns the number of bad columns. More specifically, among all strings from $E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$ the one for which $\text{bc}(S^p \cup \{s\})$ is minimal is selected. In other words, at each iteration the string that causes a minimal increase in the number of bad columns is chosen. In case of ties, the first one encountered is selected.

Algorithm 1 Greedy Algorithm

```
 1: input: set of input strings $I$, maximum number of allowed
    bad columns $k$, partial solution $S^p$
 2: if $S^p = \emptyset$ then
 3:     $s^* :=$ SelectFirstString($I$)
 4:     $S^p := \{s^*\}$
 5: end if
 6: $E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$
 7: while $E \neq \emptyset$ do
 8:     $s^* := \text{argmin}\{\text{bc}(S^p \cup \{s\}) \mid s \in E\}$
 9:     $S^p := S^p \cup \{s^*\}$
10:     $E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$
11: end while
12: output: A complete solution $S = S^p$
```

The last remaining question concerns the choice of the first string in case of an empty partial solution given as input (see function SelectFirstString(I) in line 3 of Algorithm 1). Note that, in this case, a different criterion must be used, because adding any string to the empty partial solution results in a partial solution with no bad columns at all. We decided for the following frequency-based approach. First, 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 5 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 $s_i \in I$:

$$\omega(s_i) := \sum_{j=1}^{m} \text{fr}_{j,s_i[j]} \tag{5}$$

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 in s_i. The following string is then returned by function SelectFirstString(I):

$$s := \text{argmax}\{\omega(s_i) \mid s_i \in I\} \tag{6}$$

To conclude, the advantage of this greedy algorithm is to be found in its simplicity and low resource requirements. On the downside, no performance guarantees are given.

B. Greedy-Based Pilot Method

Greedy-based pilot methods [11] are simple extensions of greedy heuristics aimed at obtaining better solutions by looking ahead at each construction step for each possible choice in order to avoid the greedy trap. These methods were initially proposed in the context of the Steiner tree problem in [12], [13]. However, in the meanwhile, applications to other problems—such as the optimization of bycicle sharing systems [14]—exist.

Greedy-based pilot methods basically work as follows. Given a current partial solution S^p, and the set E of options to extend this partial solution, the pilot method provides all partial solutions $(S^p \cup s)$, for $s \in E$, as input to the greedy algorithm which is the basic ingredient of the pilot method. In this way, $|E|$ greedy solutions are produced, and subsequently

Algorithm 2 Greedy-Based Pilot Algorithm

```
1: input: set of input strings $I$, maximum number of allowed
   bad columns $k$
2: $S^p := \emptyset$
3: $E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$
4: while $E \neq \emptyset$ do
5:     $s^* := \text{argmax}\{\mathsf{Greedy}(S^p \cup \{s\}) \mid s \in E\}$
6:     $S^p := S^p \cup \{s^*\}$
7:     $E := \{s \in I \setminus S^p \mid \text{bc}(S^p \cup \{s\}) \leq k\}$
8: end while
9: output: A complete solution $S = S^p$
```

evaluated by the objective function. The option $s^* \in E$ which is finally chosen to extend partial solution S^p is the one that led to the best greedy solution. These steps are iterated until E is the empty set, that is, until S^p is a complete solution.

Obviously, a greedy-based pilot method potentially avoids locally wrong decisions caused by the greedy function by means of the lookahead mechanism that is employed. However, on the downside, the time complexity with respect to the basic greedy algorithm increases significantly. Therefore, greedy-based pilot methods should only be employed in those cases in which the basic greedy algorithm is reasonably fast.

In the following we assume that function $\mathsf{Greedy}(S^p)$ from line 5 of Algorithm 2 calls the greedy algorithm from Algorithm 1 with partial solution S^p as input. Moreover, we assume that this function simply returns the objective function value of the complete solution constructed by Algorithm 1 with S^p as input. The greedy-based pilot method which makes use of this function is pseudo-coded in Algorithm 2.

IV. Experimental Evaluation

Henceforth, the greedy algorithm from Section III-A is denoted by GREEDY, while the greedy-based pilot method from Section III-B is denoted by PILOT. We implemented GREEDY and PILOT in ANSI C++ using GCC 4.7.3 for compiling the software. Moreover, the ILP outlined in Section II was solved with IBM ILOG CPLEX V12.1 (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. Finally, the section concludes with a detailed analysis of the experimental results.

A. 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 probabiliy of $1/|\Sigma|$ to appear at any of the m positions of s. Then, each of the n input strings is generated 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 forced at least one change per input string.

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

- $n \in \{100, 500, 1000\}$
- $m \in \{100, 500, 1000\}$
- $|\Sigma| \in \{4, 12, 20\}$
- $\mathbf{p}_c \in \{0.001, 0.003, 0.005\}$

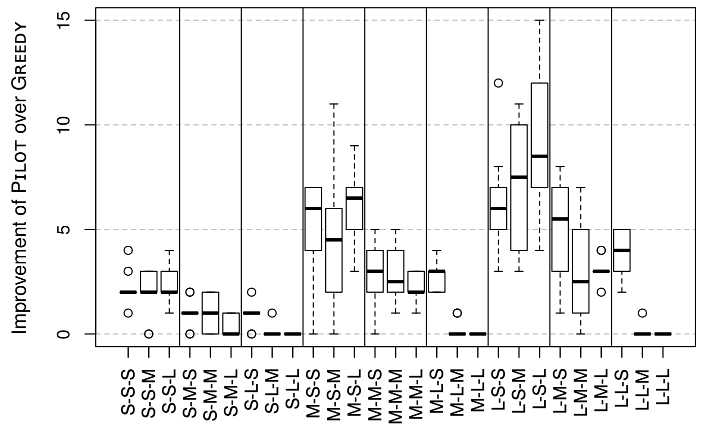
For each combination of these values we randomly generated 10 instances, which results in a total of 810 benchmark instances. To test each instance with different limits for the number of allowed bad columns, we used values for k from $\{2, n/20, n/10\}$.

B. Results

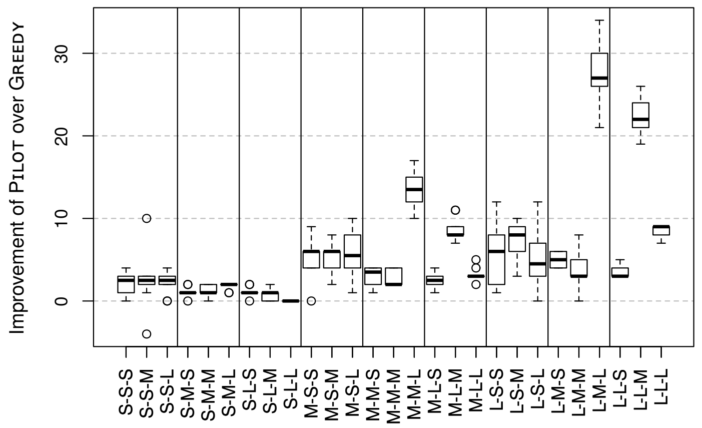
The results are presented in three Tables: Table I contains the results for all instances with $|\Sigma| = 4$, Table II contains the results for all instances with $|\Sigma| = 12$, and Table III shows the results for instances with $|\Sigma| = 20$. All three tables have the following format. The first three table columns indicate the number of input strings (n), the string length (m) and the *change probability* $(\mathbf{p}_c)$. The result of GREEDY, PILOT 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 presented in the following way. For GREEDY and PILOT with provide the average of the results 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”). For CPLEX, which was applied for a maximum of 3600 CPU seconds to each problem instance, we provide the average result (column with heading ”mean”) and the corresponding average optimality gap (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 marked in bold font.

Apart from the numerical results provided in the form of tables, the graphics of Figure 1 show the improvement of PILOT over GREED exemplary for the instances with $|\Sigma| = 20$, and the graphics of Figure 2 show the improvement of PILOT over CPLEX exemplary for the instances with $|\Sigma| = 12$. The notation X-Y-Z on the x-axis of these figures has the following meaning. X, Y, and Z take values from $\{\text{S}, \text{M}, \text{L}\}$, 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 probablity 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 0.001, M to 0.003, and L to 0.005. The following observations can be made:

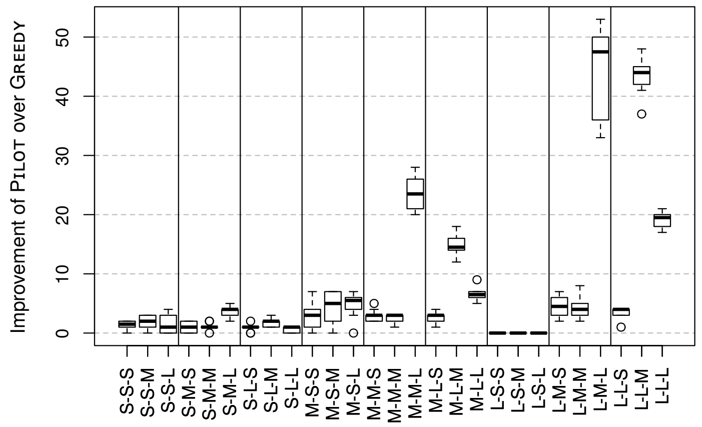
- First, no substantial differences can be observed concerning the relative performance of the algorithms for what concerns instances of different alphabet sizes. The only difference is that the objective function values decrease with increasing alphabet size.



(a) Results for $k = 2$.



(b) Results for $k = n/20$.



(c) Results for $k = n/10$.

Fig. 1. Improvement of PILOT over GREEDY (in absolute terms) concerning the instances with $|\Sigma| = 20$. Each box shows these differences for the corresponding 10 instances. Note that negative values would indicate that obtained a better result than GSA.

- Due to its design, the results of PILOT are always at least as good as those of GREEDY. Moreover, the advantages of PILOT over GREEDY tend to grow with an increasing number of input strings. Surprisingly, when $k = 2$, the advantages of PILOT over GREEDY grow with decreasing input string length. This can still, to some extent, be observed for $k = n/20$. However, when $n = n/10$ this effect is no more noticable.

- Concerning the comparison of PILOT and CPLEX we can observe that CPLEX generally outperforms PILOT for small and medium size instances. Hereby, the advantages of CPLEX over PILOT are more pronounced for what concerns medium size instances with $k = n/20$ and $k = n/10$; see Figures 2b and 2c, which show the case for instances with $|\Sigma| = 12$. On the other side, when large size instances are concerned, CPLEX is not competitive anymore.

- Concerning computation time requirements, both PILOT and CPLEX require a substantially higher computation time than GREEDY. In fact, both methods cannot be applied to instances with more than 1000 input strings.

In general, the rather big advantage of CPLEX over PILOT for medium size instances indicates that there is still a lot of room for improvement for what concerns heuristic methods.

V. CONCLUSIONS AND OUTLOOK

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. We proposed the first algorithmic approaches to solve this problem. An integer linear programming model was presented, wich was solved by CPLEX. In second place, a greedy heuristic, together with an extension known as greedy-based pilot method, was proposed. An extensive experimental evaluation revealed that the best strategy among these three approaches for instances of small and medium size is CPLEX. However, when large problem instances are concerned, CPLEX is outperformed by the greedy-based pilot method.

Concerning future work we plan to consider also other metaheuristic techniques such as, for example, local search based strategies and evolutionary algorithms. Moreover, we plan to study ways in which heuristics and metaheuristics can be beneficially combined with CPLEX for solving this problem.

TABLE I. EXPERIMENTAL RESULTS FOR INSTANCES WITH $|\Sigma| = 4$.

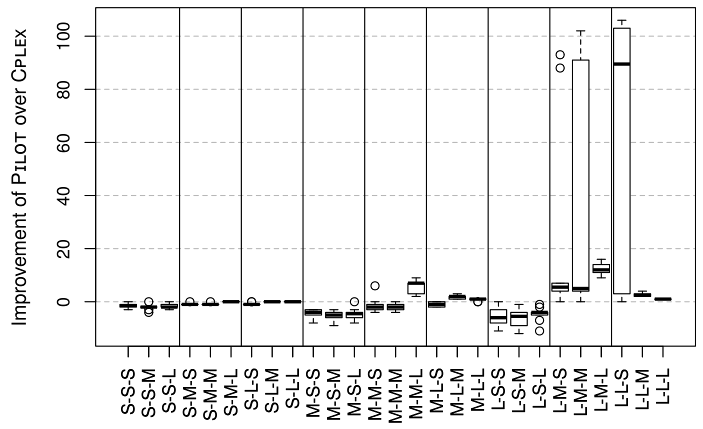
			$k = 2$						$k = n/20$						$k = n/10$					
			GREEDY		PILOT		CPLEX		GREEDY		PILOT		CPLEX		GREEDY		PILOT		CPLEX	
n	m	p_{mut}	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**gap**
		0.001	28.5	0.0	31.0	0.4	**32.1**	0.0	33.4	0.0	35.1	0.7	**39.9**	0.0	41.2	0.0	42.8	1.7	**50.4**	0.0
	100	0.003	30.2	0.0	31.9	0.5	**33.2**	0.0	35.3	0.0	37.0	0.7	**40.1**	0.0	43.0	0.0	44.8	1.6	**50.2**	0.0
		0.005	27.6	0.0	29.4	0.4	**31.5**	0.0	33.8	0.0	35.9	0.7	**39.6**	0.0	41.8	0.0	43.8	1.4	**49.9**	0.0
		0.001	29.1	0.0	30.1	1.6	**31.1**	0.0	32.3	0.0	33.1	2.6	**35.4**	0.0	37.6	0.0	38.5	7.0	**40.5**	0.0
100	500	0.003	28.1	0.0	29.2	1.5	**30.0**	0.0	31.4	0.0	32.5	2.7	**34.6**	0.0	37.1	0.0	38.1	6.1	**40.7**	0.0
		0.005	9.1	0.0	10.6	0.6	**11.0**	0.0	11.3	0.0	13.9	1.8	**14.5**	0.0	14.6	0.0	19.1	4.9	**19.5**	0.0
		0.001	27.7	0.0	28.6	2.5	**29.3**	0.0	31.2	0.0	31.9	5.2	**33.3**	0.0	36.4	0.0	37.1	10.9	**38.5**	0.0
	1000	0.003	3.6	0.0	**4.4**	0.6	**4.4**	0.0	4.8	0.0	**7.4**	2.3	**7.4**	0.0	6.9	0.0	**12.4**	7.9	**12.4**	0.0
		0.005	**1.6**	0.0	**1.6**	0.4	**1.6**	0.0	2.4	0.0	**3.0**	0.6	**3.0**	0.0	3.8	0.0	**5.7**	2.0	**5.7**	0.0
		0.001	135.4	0.0	139.6	29.4	**144.5**	0.0	243.0	0.0	247.1	260.3	**284.8**	0.0	348.5	0.0	351.4	743.9	**391.8**	0.0
	100	0.003	136.7	0.0	141.0	29.6	**144.5**	0.0	238.6	0.0	244.0	247.8	**282.2**	0.0	345.9	0.0	348.7	695.8	**388.4**	0.0
		0.005	128.2	0.0	133.3	33.9	**137.8**	0.0	238.2	0.0	242.6	247.8	**279.3**	0.0	344.7	0.0	349.0	693.1	**387.6**	0.0
		0.001	126.0	0.0	129.2	122.8	**131.3**	3.7	164.4	0.0	167.4	1016.6	**198.1**	0.7	206.6	0.0	208.9	3291.8	**248.2**	0.0
500	500	0.003	131.3	0.0	133.6	107.8	**135.5**	0.0	170.5	0.0	172.6	1015.7	**202.6**	0.0	212.4	0.0	214.6	3285.6	**252.6**	0.0
		0.005	36.4	0.0	**39.4**	40.3	22.8	>99.9	60.0	0.0	**71.0**	793.9	31.2	>99.9	84.2	0.0	105.1	2830.0	**117.7**	70.0
		0.001	130.3	0.1	132.6	176.2	**134.2**	0.0	162.0	0.1	164.0	1785.3	**184.6**	3.1	195.0	0.1	196.9	6031.1	**233.5**	0.0
	1000	0.003	6.9	0.1	**8.2**	28.3	1.1	>99.9	18.0	0.0	**32.8**	1472.0	0.2	>99.9	32.6	0.1	**59.2**	4243.6	3.8	>99.9
		0.005	2.2	0.1	**2.4**	21.4	0.0	>99.9	8.0	0.0	**18.5**	518.6	0.0	>99.9	14.9	0.0	**32.4**	1676.1	0.0	>99.9
		0.001	270.4	0.0	276.9	215.7	**282.6**	0.0	664.3	0.0	668.8	4093.4	**733.0**	0.0	**1000.0**	0.0	**1000.0**	5452.1	**1000.0**	0.0
	100	0.003	271.4	0.0	276.1	227.0	**280.5**	0.0	660.6	0.0	666.1	3052.9	**735.9**	0.0	**1000.0**	0.0	**1000.0**	5888.8	**1000.0**	0.0
		0.005	265.8	0.0	274.4	225.0	**280.4**	0.0	666.0	0.0	671.9	3207.4	**737.1**	0.0	**1000.0**	0.0	**1000.0**	5965.6	**1000.0**	0.0
		0.001	257.7	0.1	**260.9**	756.6	29.8	>99.9	372.0	0.1	375.8	16847.3	**397.8**	>99.9	487.7	0.1	491.0	56021.0	**529.3**	>99.9
1000	500	0.003	254.0	0.1	**258.2**	743.4	24.3	>99.9	372.0	0.1	375.9	15575.8	**399.3**	>99.9	485.6	0.1	488.9	53951.5	**586.1**	0.0
		0.005	63.6	0.1	**69.0**	244.1	6.6	>99.9	134.5	0.1	**162.0**	15158.4	0.3	>99.9	221.7	0.1	**261.2**	48604.2	0.0	>99.9
		0.001	253.0	0.1	**256.3**	1858.2	127.9	>99.9	334.0	0.1	**337.8**	27098.1	111.8	>99.9	422.3	0.1	**425.4**	98938.2	0.0	>99.9
	1000	0.003	17.7	0.1	**20.0**	225.0	0.0	>99.9	45.5	0.1	**76.4**	21440.4	0.0	>99.9	78.7	0.1	**134.5**	73282.7	0.0	>99.9
		0.005	3.0	0.1	**3.6**	137.7	0.0	>99.9	16.3	0.1	**42.3**	12758.5	0.0	>99.9	32.8	0.1	**75.1**	31936.3	0.0	>99.9

TABLE II. Experimental results for instances with $|\Sigma| = 12$.

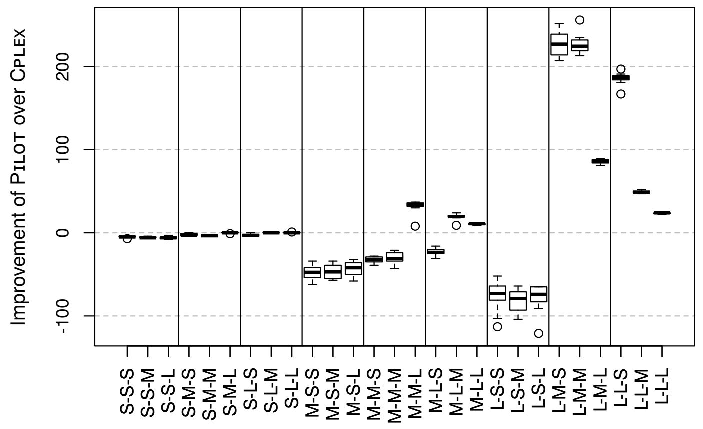
			$k = 2$						$k = n/20$						$k = n/10$					
			Greedy		Pilot		Cplex		Greedy		Pilot		Cplex		Greedy		Pilot		Cplex	
n	m	p_{mut}	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**gap**
		0.001	10.3	0.0	12.2	0.3	**13.7**	0.0	16.4	0.0	18.1	0.6	**22.8**	0.0	26.6	0.0	28.1	1.5	**35.8**	0.0
	100	0.003	12.3	0.0	14.5	0.3	**16.6**	0.0	17.7	0.0	20.3	0.6	**25.6**	0.0	26.1	0.0	28.5	1.5	**37.3**	0.0
		0.005	11.5	0.0	14.3	0.3	**16.1**	0.0	17.4	0.0	19.0	0.5	**24.8**	0.0	26.7	0.0	28.3	1.4	**35.6**	0.0
		0.001	9.8	0.0	10.8	0.8	**11.7**	0.0	13.5	0.0	14.8	2.2	**17.1**	0.0	19.4	0.0	20.6	5.6	**24.0**	0.0
100	500	0.003	11.6	0.0	12.7	0.8	**13.6**	0.0	15.2	0.0	16.1	2.0	**19.6**	0.0	21.0	0.0	22.1	5.6	**27.1**	0.0
		0.005	2.2	0.0	**3.0**	0.3	**3.0**	0.0	3.8	0.0	6.0	1.1	**6.1**	0.0	6.5	0.0	**11.1**	3.5	**11.1**	0.0
		0.001	10.8	0.0	11.9	1.4	**12.7**	0.0	14.3	0.0	15.1	3.6	**17.8**	0.0	20.3	0.0	21.1	10.5	**23.1**	0.0
	1000	0.003	1.7	0.0	**1.8**	0.4	**1.8**	0.0	2.8	0.0	**3.7**	0.8	**3.7**	0.0	4.6	0.0	6.4	2.5	**6.6**	0.0
		0.005	**1.0**	0.0	**1.0**	0.4	**1.0**	0.0	1.3	0.0	**1.4**	0.4	1.3	>99.9	2.4	0.0	**3.1**	0.8	**3.1**	>99.9
		0.001	47.0	0.0	52.8	21.6	**57.1**	0.0	173.0	0.0	177.6	235.1	**226.0**	0.0	305.1	0.0	309.6	762.0	**356.2**	0.0
	100	0.003	49.0	0.0	54.2	20.4	**59.6**	0.0	174.9	0.0	178.3	267.0	**224.7**	0.0	305.9	0.0	310.4	747.1	**352.5**	0.0
		0.005	54.1	0.0	58.7	20.6	**63.4**	0.0	179.4	0.0	183.2	246.9	**225.7**	0.0	310.2	0.0	314.1	847.9	**356.3**	0.0
		0.001	46.2	0.0	49.4	49.6	**50.7**	>99.9	91.8	0.0	94.7	1255.5	**126.9**	28.5	138.5	0.0	141.0	3479.9	**184.7**	12.9
500	500	0.003	44.6	0.0	47.9	63.7	**50.0**	>99.9	91.3	0.0	94.1	1045.2	**124.1**	33.3	135.3	0.0	138.1	3290.0	**181.5**	14.2
		0.005	4.3	0.0	**7.1**	24.0	1.5	>99.9	22.3	0.0	**35.1**	637.7	3.6	>99.9	42.1	0.0	**66.7**	2265.1	65.2	>99.9
		0.001	45.2	0.1	47.8	105.3	**48.9**	>99.9	78.5	0.1	80.5	1677.7	**103.8**	22.7	113.3	0.1	115.7	5761.4	**154.3**	11.1
	1000	0.003	2.0	0.1	**2.1**	21.4	0.3	>99.9	10.4	0.1	**20.3**	679.0	1.2	>99.9	20.1	0.1	**37.0**	2253.2	0.2	>99.9
		0.005	**1.0**	0.1	**1.0**	22.0	0.2	>99.9	6.3	0.1	**10.4**	189.5	0.0	>99.9	11.9	0.1	**20.5**	730.1	0.0	>99.9
		0.001	105.5	0.0	110.8	138.3	**116.2**	18.6	575.8	0.0	582.7	3094.1	**658.5**	0.0	**1000.0**	0.0	**1000.0**	6242.1	**1000.0**	0.0
	100	0.003	100.2	0.0	107.4	140.3	**113.4**	19.1	572.9	0.0	578.9	2927.3	**660.5**	0.0	**1000.0**	0.0	**1000.0**	5202.9	**1000.0**	0.0
		0.005	105.4	0.0	110.5	143.2	**115.2**	19.1	576.8	0.0	582.3	3264.7	**660.8**	0.0	**1000.0**	0.0	**1000.0**	6316.5	**1000.0**	0.0
		0.001	88.3	0.1	**93.9**	372.0	72.2	>99.9	222.0	0.1	**226.9**	5651.2	0.0	>99.9	355.7	0.1	360.5	21149.8	**460.8**	10.5
1000	500	0.003	91.3	0.1	**95.5**	514.3	55.8	>99.9	223.5	0.1	**227.0**	15932.1	0.0	>99.9	358.1	0.1	**361.9**	59007.8	320.9	>99.9
		0.005	9.0	0.1	**12.3**	169.5	0.0	>99.9	59.0	0.1	**85.7**	10722.1	0.0	>99.9	128.5	0.1	**171.1**	40490.8	0.0	>99.9
		0.001	91.6	0.2	**95.0**	502.0	34.8	>99.9	181.6	0.2	**185.4**	28734.5	0.0	>99.9	272.5	0.2	**275.7**	104883.3	0.0	>99.9
	1000	0.003	2.1	0.2	**2.7**	143.8	0.0	>99.9	21.4	0.2	**49.2**	14109.5	0.0	>99.9	44.4	0.2	**93.6**	47100.0	0.0	>99.9
		0.005	**1.0**	0.2	**1.0**	146.3	0.0	>99.9	12.4	0.2	**23.7**	3837.5	0.0	>99.9	24.2	0.2	**47.9**	156310.2	0.0	>99.9

TABLE III. EXPERIMENTAL RESULTS FOR INSTANCES WITH $|\Sigma| = 20$.

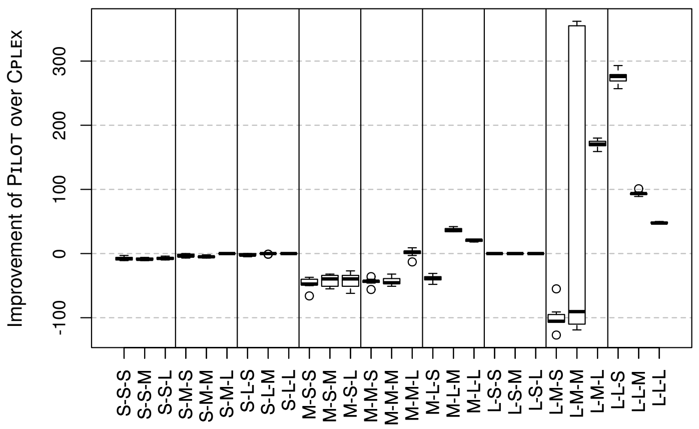
			$k = 2$						$k = n/20$						$k = n/10$					
			GREEDY		PILOT		CPLEX		GREEDY		PILOT		CPLEX		GREEDY		PILOT		CPLEX	
n	m	p_{mut}	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**gap**	**mean**	**time**	**mean**	**time**	**mean**	**gap**
		0.001	9.1	0.0	11.3	0.3	**12.9**	0.0	14.8	0.0	16.8	0.6	**21.9**	0.0	24.4	0.0	25.8	1.4	**34.0**	0.0
	100	0.003	9.3	0.0	11.3	0.2	**12.9**	0.0	14.4	0.0	16.9	0.6	**22.1**	0.0	23.9	0.0	25.8	1.4	**34.1**	0.0
		0.005	8.4	0.0	10.7	0.2	**12.3**	0.0	13.7	0.0	15.9	0.5	**21.5**	0.0	23.6	0.0	25.2	1.3	**33.7**	0.0
		0.001	9.5	0.0	10.5	0.8	**11.7**	0.0	13.5	0.0	14.6	2.1	**17.7**	0.0	19.5	0.0	20.5	5.7	**25.8**	0.0
100	500	0.003	7.0	0.0	8.0	0.7	**8.6**	0.0	10.7	0.0	11.9	1.9	**14.6**	0.0	16.4	0.0	17.5	5.2	**23.2**	0.0
		0.005	2.0	0.0	**2.3**	0.3	**2.3**	0.0	3.4	0.0	**5.2**	0.9	**5.2**	0.0	6.2	0.0	**9.9**	3.1	**9.9**	0.0
		0.001	7.5	0.0	8.4	1.5	**9.2**	0.0	10.5	0.0	11.6	3.6	**13.5**	0.0	16.0	0.0	16.9	9.7	**18.7**	0.0
	1000	0.003	1.3	0.0	**1.4**	0.4	**1.4**	0.0	2.3	0.0	**3.0**	0.7	**3.0**	0.0	4.0	0.0	**5.8**	2.2	**5.8**	0.0
		0.005	**1.0**	0.0	**1.0**	0.4	**1.0**	0.0	**1.1**	0.0	**1.1**	0.4	**1.1**	>99.9	2.1	0.0	**2.8**	0.9	2.7	>99.9
		0.001	38.4	0.0	43.4	19.0	**48.0**	0.0	170.0	0.0	176.2	236.9	**215.6**	0.0	298.9	0.0	301.8	756.0	**348.8**	0.0
		0.003	37.8	0.0	42.2	19.4	**45.8**	0.0	165.4	0.0	170.6	256.5	**215.2**	0.0	296.6	0.0	300.8	775.2	**348.7**	0.0
		0.005	35.4	0.0	41.6	20.4	**46.2**	0.0	163.8	0.0	169.3	255.1	**213.8**	0.0	292.6	0.0	297.4	768.5	**345.7**	0.0
		0.001	29.3	0.1	32.3	45.8	**33.4**	>99.9	74.4	0.1	77.5	1018.7	**105.5**	56.6	121.2	0.1	124.1	4300.9	**170.3**	19.7
		0.003	26.5	0.1	29.2	39.9	**30.1**	>99.9	71.4	0.1	74.1	1000.8	**94.4**	>99.9	117.7	0.1	120.2	3383.9	**164.8**	24.4
		0.005	2.9	0.1	**5.1**	21.5	1.5	>99.9	19.1	0.1	**32.7**	696.1	8.7	>99.9	39.0	0.1	**62.5**	2253.9	55.6	>99.9
		0.001	28.4	0.1	**31.3**	64.4	28.7	>99.9	61.8	0.1	64.3	1672.8	**83.4**	>99.9	99.0	0.1	101.6	5796.4	**140.5**	27.9
		0.003	1.8	0.1	**2.0**	22.3	0.2	>99.9	9.9	0.1	**18.6**	545.1	0.1	>99.9	19.0	0.1	**33.9**	1801.6	0.0	>99.9
		0.005	**1.0**	0.1	**1.0**	22.2	0.2	>99.9	6.0	0.1	**9.2**	157.3	0.0	>99.9	11.5	0.1	**18.2**	558.2	0.0	>99.9
		0.001	71.3	0.0	77.6	222.1	**82.9**	24.4	560.0	0.0	567.1	2988.1	**645.0**	0.0	**1000.0**	0.0	**1000.0**	5677.3	**1000.0**	0.0
		0.003	68.1	0.0	75.3	214.7	**80.9**	76.5	558.3	0.0	568.0	2577.2	**646.7**	0.0	**1000.0**	0.0	**1000.0**	6735.0	**1000.0**	0.0
		0.005	66.9	0.0	76.9	210.9	**83.6**	21.4	561.4	0.0	568.4	4330.0	**648.2**	0.0	**1000.0**	0.0	**1000.0**	9004.8	**1000.0**	0.0
		0.001	55.7	0.1	**60.9**	618.8	16.0	>99.9	188.2	0.1	**193.3**	16834.8	29.4	>99.9	323.9	0.1	**328.3**	59750.2	208.5	>99.9
		0.003	53.4	0.1	**57.2**	706.3	15.5	>99.9	189.9	0.1	**193.6**	15733.9	0.0	>99.9	324.7	0.1	328.9	55671.2	**339.3**	>99.9
		0.005	4.7	0.1	**7.7**	233.5	0.0	>99.9	47.7	0.1	**74.9**	10357.7	0.0	>99.9	111.1	0.1	**156.2**	37409.2	0.0	>99.9
		0.001	54.1	0.2	**58.0**	415.0	17.8	>99.9	147.2	0.2	**150.7**	30391.4	0.0	>99.9	244.6	0.2	**248.0**	110501.7	0.0	>99.9
		0.003	2.0	0.2	**2.1**	142.4	0.0	>99.9	19.6	0.2	**41.8**	10466.8	0.0	>99.9	40.3	0.2	**83.8**	41936.1	0.0	>99.9
		0.005	1.0	0.2	**1.0**	146.3	0.0	>99.9	11.9	0.2	**20.3**	2792.5	0.0	>99.9	23.7	0.2	**43.0**	12057.9	0.0	>99.9



(a) Results for $k = 2$.



(b) Results for $k = n/20$.



(c) Results for $k = n/10$.

Fig. 2. Improvement of PILOT over CPLEX (in absolute terms) concerning the instances with $|\Sigma| = 12$. Each box shows these differences for the corresponding 10 instances. Note that negative values indicate that CPLEX obtained a better result than PILOT.

Acknowledgement. This work was supported by project TIN2012-37930 (Spanish Government) and project SGR 2014-1034 (Generalitat de Catalunya). Moreover, support is acknowledged from IKERBASQUE. Finally, Evelia Lizárraga-Olivas acknowledges support by a doctoral grant (number 253787) from the Mexican National Council for Science and Technology (CONACYT). All experiments were executed in the High Performance Computing environment managed by RDlab (http://rdlab.cs.upc.edu) and we thank them for their support.

REFERENCES

[1] C. Meneses, C. Oliveira, and P. Pardalos, “Optimization techniques for string selection and comparison problems in genomics,” *IEEE Engineering in Medicine and Biology Magazine*, vol. 24, no. 3, pp. 81–87, 2005.

[2] S. Mousavi, M. Babaie, and M. Montazerian, “An improved heuristic for the far from most strings problem,” *Journal of Heuristics*, vol. 18, pp. 239–262, 2012.

[3] E. Pappalardo, P. M. Pardalos, and G. Stracquadanio, *Optimization approaches for solving string selection problems*. Springer New York, 2013.

[4] W. J. Hsu and M. W. Du, “Computing a longest common subsequence for a set of strings,” *BIT Numerical Mathematics*, vol. 24, no. 1, pp. 45–59, 1984.

[5] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[6] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, ser. Computer Science and Computational Biology. Cambridge University Press, Cambridge, 1997.

[7] S. Rajasekaran, H. Nick, P. M. Pardalos, S. Sahni, and G. Shaw, “Efficient algorithms for local alignment search,” *Journal of Combinatorial Optimization*, vol. 5, no. 1, pp. 117–124, 2001.

[8] S. Rajasekaran, Y. Hu, J. Luo, H. Nick, P. M. Pardalos, S. Sahni, and G. Shaw, “Efficient algorithms for similarity search,” *Journal of Combinatorial Optimization*, vol. 5, no. 1, pp. 125–132, 2001.

[9] C. Boucher, G. M. Landau, A. Levy, D. Pritchard, and O. Weimann, “On approximating string selection problems with outliers,” *Theoretical Computer Science*, vol. 498, pp. 107–114, 2013.

[10] G. M. Landau, J. P. Schmidt, and D. Sokol, “An algorithm for approxixmate tandem repeat,” *Journal of Computational Biology*, vol. 8, no. 1, pp. 1–18, 2001.

[11] S. Voß, A. Fink, and C. Duin, “Looking ahead with the pilot method,” *Annals of Operations Research*, vol. 136, no. 1, pp. 285–302, 2005.

[12] C. Duin and S. Voß, “Steiner tree heuristicsA survey,” in *Operations Research Proceedings 1993*. Springer Verlag, 1994, pp. 485–496.

[13] ——, “The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs,” *Networks*, vol. 34, no. 3, pp. 181–191, 1999.

[14] M. Rainer-Harbach, P. Papazek, G. R. Raidl, B. Hu, and C. Kloimüllner, “Pilot, grasp, and vns approaches for the static balancing of bicycle sharing systems,” *Journal of Global Optimization*, 2014, in press.