INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 27 (2020) 91–111
DOI: 10.1111/itor.12603

Minimum common string partition: on solving large-scale problem instances

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, Carrer de Can Planas, 08193 Bellaterra, Spain
E-mail: christian.blum@iiia.csic.es [Blum]

Received 8 February 2018; received in revised form 9 July 2018; accepted 20 September 2018

Abstract

Minimum common string partition is an NP-hard combinatorial optimization problem from the bioinformatics field. The current state-of-the-art algorithm is a hybrid technique known as construct, merge, solve, and adapt (CMSA). This algorithm combines two main algorithmic components: generating solutions in a probabilistic way and solving reduced subinstances obtained from the tackled problem instances, if possible, to optimality. However, the CMSA algorithm was not intended for application to very large problem instances. Therefore, in this paper we present a technique that makes CMSA, and other available algorithms for this problem, applicable to problem instances that are about one order of magnitude larger than the largest problem instances considered so far. Moreover, a reduced variable neighborhood search (RVNS) for solving the tackled problem, based on integer programming, is introduced. The experimental results show that the modified CMSA algorithm is very strong for problem instances based on rather small alphabets. With growing alphabet size, it turns out that RVNS has a growing advantage over CMSA.

Keywords: minimum common string partition; large-scale problem instances; reduced variable neighborhood search

1. Introduction

Optimization problems based on strings are abundant in research fields such as bioinformatics (Gusfield, 1997; Blum and Festa, 2016) and text processing (Manning et al., 2008). In the context of computer science, a string s is a data type for representing and storing sequence information in terms of a finite sequence of characters from a (generally finite) alphabet Σ. Each string s has a length denoted by $|s|$. Words, and even whole texts, may be stored by means of strings. Strings also play an important role in bioinformatics because most of the genetic instructions involved in the growth, development, functioning, and reproduction of living organisms are stored in deoxyribonucleic acid (DNA) molecules, which can be represented by strings over the alphabet $\Sigma = \{A, C, T, G\}$.

Similarly, most proteins can be stored as strings of letters from an alphabet of size 20, representing the 20 standard amino acids that are the building blocks of most proteins.

The minimum common string partition (MCSP) problem is a string-based combinatorial optimization problem from the bioinformatics field. The problem input consists of two strings, s_1 and s_2. Both input strings are composed of letters from a finite alphabet Σ. Moreover, they fulfill the property of being *related*, which means that each letter of Σ has the same number of occurrences in each input string. The property of being related implies that s_1 and s_2 have the same length n, that is, $|s_1| = |s_2| = n$. A candidate solution to the MCSP problem is obtained by cutting s_1, respectively, s_2, into pieces, resulting in a set P_1, respectively, P_2, of nonoverlapping substrings. A candidate solution (P_1, P_2) is a valid solution, if $P_1 = P_2$. The optimization goal consists in finding a valid solution (P_1, P_2) that minimizes $|P_1|$.[1] As an example, consider the following two DNA sequences: $s_1 =$ **AAGACTG** and $s_2 =$ **ACTAGGA**. Counting the number of the occurrences of all letters in both strings, it is easy to confirm that these two strings are related. A trivial valid solution can be obtained for any problem instance by cutting both input strings into substrings of length 1. In the case of the above-mentioned example, the trivial valid solution is $P_1 = P_2 = \{\mathbf{A}, \mathbf{A}, \mathbf{A}, \mathbf{C}, \mathbf{T}, \mathbf{G}, \mathbf{G}\}$, with an objective function value of 7. Note that the optimal solution in this example is $P_1 = P_2 = \{\mathbf{ACT}, \mathbf{AG}, \mathbf{G}, \mathbf{A}\}$, with objective function value 4. The MCSP problem, as pointed out by Chen et al. (2005), is closely related to the problem of sorting by reversals with duplicates, which is one of the key problems in genome rearrangement. It has been shown to be NP-hard even in very restrictive cases (Goldstein et al., 2005). Moreover, Jiang et al. (2012) proved the NP-completeness of the decision version of the MCSP^c problem—with c being the alphabet size—for $c \geq 2$.

1.1. Existing works

Research work on the MCSP began with the development of approximation algorithms (see, e.g., Shapira and Storer, 2002; Chrobak et al., 2004; Kolman, 2005; Cormode and Muthukrishnan, 2007; Kolman and Waleń, 2007). More recently, Goldstein and Lewenstein (2011) proposed a greedy algorithm that runs in $O(n)$ time. Another greedy algorithm obtaining better results (on average) was presented in He (2007). The first metaheuristics proposed in the related literature for the MCSP problem were (a) the Min-Max Ant System from Ferdous and Sohel Rahman (2013, 2017) and (b) the probabilistic tree search algorithm from Blum et al. (2014). Both works used a range of artificial and real DNA instances from Ferdous and Sohel Rahman (2013) for the experimental evaluation. However, it was discovered that the best results, at least for the smaller problem instances used in Ferdous and Sohel Rahman (2013), are obtained by integer linear programming (ILP) techniques. The first ILP model was proposed in Blum et al. (2015), together with a deterministic 2-phase heuristic based on the ILP model. Later on, improved ILP models were presented in Ferdous and Sohel Rahman (2015) and Blum and Raidl (2016). The current state-of-the-art technique is a construct, merge, solve, and adapt (CMSA) approach from Blum et al. (2016). This hybrid technique sequentially applies an ILP solver to a reduced subinstance of the original problem instance. The CMSA variant from Blum et al. (2016) is based on the ILP model from Blum et al. (2015).

[1]Note that minimizing $|P_1|$ is similar to minimizing $|P_2|$.

1.2. Our contribution

The main problem with CMSA, the current state-of-the-art technique, is that the number of binary variables used in the available ILP models is very large and grows exponentially with the input string length. The largest problem instances studied in Blum et al. (2016) consist of input strings of length 2000. In order to make CMSA, as well as the available ILP models, applicable to much larger problem instances, we first present a modification that allows a substantial reduction in the size of the models, while still being able to obtain optimal solutions to the tackled problem instances. Second, after applying the modified models to problem instances whose size—in terms of the input string length—ranges from 2000 to 20,000, we use the obtained results in order to decide for one of the modified ILP models for the application within CMSA. Moreover, we develop an ILP-based reduced variable neighborhood search (RVNS) technique, which uses the same modified ILP model. An extensive experimental evaluation shows that CMSA has advantages over RVNS for instances based on rather small alphabets, while the opposite is the case for instances based on rather large alphabets. Finally, note that a preliminary version of this work was published in the conference proceedings of variable neighborhood search (VNS) 2017 (Blum, 2018). The extension concerns the following aspects. In Blum (2018), we only provided a straightforward modification of the ILP model from Blum et al. (2015) and the corresponding CMSA algorithm. In this work, we test the same modification on two different ILP models and choose the better one—according to the obtained results—for its use within CMSA and RVNS. Moreover, we make both algorithms applicable to large problem instances based on small alphabets. In particular, the algorithm versions from Blum (2018) were not applicable to instances with input strings of length 20,000 based on an alphabet of size 4, for example. Finally, the experimental evaluation is extended by considering instances of a wider range of alphabet sizes.

1.3. Organization of this paper

The remainder of this paper is structured as follows. The modification of the existing ILP models is described in Section 2. Moreover, this section provides a detailed experimental evaluation of the original models and the modified models on a wide range of problem instances. The redesigned CMSA algorithm applicable to large problem instances is described in Section 3, whereas the ILP-based RVNS algorithm is detailed in Section 4. Finally, an experimental evaluation of both CMSA and RVNS, in comparison to the standard greedy algorithm and the best results obtained from solving the ILP models, is provided in Section 5. The paper concludes with a summary of the findings and an outlook to future work.

2. Modification of the ILP models

This section deals with the crucial idea of this work: it turns out to be possible to define modified ILP models whose optimal solutions, which are partial solutions with respect to the original problem instances, can be transformed in a straightforward way—and with a linear time complexity—into optimal solutions to the original problem instances. First, the modification of the ILP from Blum

et al. (2015), known as ILP_{cb}, is presented. Second, the corresponding modification of the alternative ILP presented in Blum and Raidl (2016), known as ILP_{cs}, is described. Finally, both original and modified models are applied to a wide range of problem instances. Note that the third existing ILP model from Ferdous and Sohel Rahman (2015) is not considered here because a corresponding modification is not as straightforward as for the other two models.

2.1. *Modification of* ILP_{cb}

Model ILP_{cb} is based on the concept of *common blocks*. Each common block b_i of input strings s_1 and s_2 is a triple $(t_i, k1_i, k2_i)$ consisting of a string t_i and two indexes $0 \leq k1_i \leq n$ and $0 \leq k2_i \leq n$. Hereby, t_i is a substring of both s_1 and s_2. Moreover, t_i starts at position $k1_i$ in s_1, while it starts at position $k2_i$ in s_2. Henceforth, let B denote the set of all possible common blocks with respect to s_1 and s_2. Using this definition, the MCSP problem can be rephrased as a subset problem, that is, any solution S is represented by a subset of common blocks from B, $S \subset B$. A candidate solution S has to fulfill the following conditions in order to be a valid solution:

1. $\sum_{b_i \in \mathcal{S}} |t_i| = n$, that is, the sum of the length of the substrings corresponding to the common blocks in S is equal to the length of the input strings.
2. For any two common blocks $b_i \neq b_j \in \mathcal{S}$, it holds that the substrings they represent neither overlap in s_1 nor in s_2.

The goal is to find a valid solution S^* such that $|S^*|$ is minimal. The optimal solution S^* for the example instance from Section 1, for example, can be represented as $S^* = \{(\mathbf{ACT}, 4, 1), (\mathbf{AG}, 2, 4), (\mathbf{G}, 7, 6), (\mathbf{A}, 1, 7)\}$.

Model ILP_{cb} uses for each common block $b_i \in B$ a binary variable x_i indicating if it is selected for the solution:

$$\min \quad \sum_{i=1}^{|B|} x_i \tag{1}$$

$$\text{s.t.} \quad \sum_{b_i \in B \text{ s.t. } k1_i \leq j < k1_i + |t_i|} x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{2}$$

$$\sum_{b_i \in B \text{ s.t. } k2_i \leq j < k2_i + |t_i|} x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{3}$$

$$x_i \in \{0, 1\} \quad \text{for } b_i \in B.$$

The objective function (1) counts the number of selected common blocks. Equation (2), respectively, Equation (3), ensures that each position $j = 1, \ldots, n$ of string s_1, respectively, string s_2, is (1) covered by exactly one selected common block and that (2) selected common blocks do not overlap. Note that Equations (2) and (3) implicitly ensure that the sum of the lengths of the selected common blocks is n.

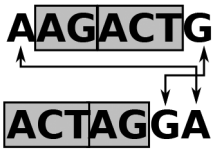


Fig. 1. Input sequences: s_1 = **AAGACTG**, s_2 = **ACTAGGA**. Solving model $\text{ILP}_{\text{cb}}^{\text{mod}}$ has resulted in the partial solution containing the common blocks that are gray-shaded. In order to obtain a complete optimal solution for the problem instance, the first uncovered **A** in s_1 is matched with the first uncovered **A** in s_2. Moreover, the first uncovered **G** in s_1 is matched with the first uncovered **G** in s_2. The complete optimal ILP_{cb} solution is, thus, $S^* = \{(\textbf{ACT}, 4, 1), (\textbf{AG}, 2, 4), (\textbf{A}, 1, 7), (\textbf{A}, 7, 6)\}$.

Note that the size of B is a determining factor for the computational difficulty of solving ILP_{cb}. In particular, $|B|$ is exponential in the length of the input strings. A problem instance with $n = 1000$ and $|\Sigma| = 4$, for example, results already in $|B| \approx 334.000$, that is, model ILP_{cb} has already approximately 334.000 variables. Moreover, in Blum et al. (2015) it was shown that a large percentage of the common blocks of B contain substrings of length 1. In the case of randomly generated instances with $n = 1000$ and $|\Sigma| = 4$, for example, this is the case for about 75% of all common blocks. If it were possible to solve the problem to optimality without implicitly regarding the blocks with substrings of size 1, it would be possible to solve problem instances with much longer input strings. Indeed, this turns out to be possible, in the following way. First, let $B^{>1} \subset B$ be the set of common blocks with substrings consisting of at least two letters. The modified ILP, henceforth denoted by $\text{ILP}_{\text{cb}}^{\text{mod}}$, is obtained as follows:

1. We replace all occurrences of B with $B^{>1}$.
2. We use a new objective function to minimize: $\sum_{b_i \in B^{>1}} x_i + (n - \sum_{b_i \in B^{>1}} x_i |t_i|)$
3. We exchange the equality signs in Equations (2) and (3) with a "$\leq$" symbol.

Note that the new objective function minimizes the sum of (a) the number of common blocks selected from $B^{>1}$ and (b) the number of uncovered positions, given the set of selected common blocks. Obviously, an optimal solution to $\text{ILP}_{\text{cb}}^{\text{mod}}$ is a partial solution with respect to the tackled MCSP problem instance. In the case of the example from Section 1, the optimal solution of $\text{ILP}_{\text{cb}}^{\text{mod}}$ contains common blocks (**ACT**, 4, 1) and (**AG**, 2, 4). And, as this results in two uncovered positions in each input string, the objective function value of this $\text{ILP}_{\text{cb}}^{\text{mod}}$ solution is 4. On the basis of an optimal $\text{ILP}_{\text{cb}}^{\text{mod}}$ solution, an optimal ILP_{cb} solution can easily be obtained by adding the correct common blocks from $B \setminus B^{>1}$—that is, common blocks with substrings of length 1—to the solution. These blocks are identified by finding a bijective mapping from the uncovered positions of s_1 to the uncovered positions of s_2, in a way such that positions that are mapped to each other have the same letters. This can be done in $O(n)$ time, simply by mapping the first uncovered occurrence of each letter $l \in \Sigma$ in s_1 to the first uncovered occurrence of this letter in s_2, the second occurrence of this letter in s_1 to the second one in s_2, and so on. See Fig. 1 for how this would be done in the case of the example of Section 1.

It is easy to see that a solution S^* of ILP_{cb} obtained from an optimal solution $S^{*,\text{mod}}$ of $\text{ILP}_{\text{cb}}^{\text{mod}}$, in the way as described above, is an optimal solution. Assume that S^* would not be an optimal solution of ILP_{cb}. This would mean that there exists at least one solution S' of ILP_{cb} with $|S'| <$

$|S^*|$. Obviously, it holds that $|S^*| = |S^{*,\text{mod}}| + |S^* \setminus S^{*,\text{mod}}|$. Moreover, as S' can be transformed into a solution $S^{\prime,\text{mod}}$ of $\text{ILP}_{\text{cb}}^{\text{mod}}$ by removing all the blocks with substrings of size 1, it holds that $|S'| = |S^{\prime,\text{mod}}| + |S' \setminus S^{\prime,\text{mod}}|$. Because $|S'| < |S^*|$, it must hold that $|S^{\prime,\text{mod}}| + |S' \setminus S^{\prime,\text{mod}}| < |S^{*,\text{mod}}| + |S^* \setminus S^{*,\text{mod}}|$. On the other hand, as $S^{*,\text{mod}}$ is an optimal solution of $\text{ILP}_{\text{cb}}^{\text{mod}}$, it must hold that $|S^{\prime,\text{mod}}| + |S' \setminus S^{\prime,\text{mod}}| \geq |S^{*,\text{mod}}| + |S^* \setminus S^{*,\text{mod}}|$. This is a contradiction to the previous statement. Therefore, the procedure described above transforms an optimal solution of $\text{ILP}_{\text{cb}}^{\text{mod}}$ into an optimal solution of ILP_{cb}.

2.2. *Modification of* ILP_{cs}

Model ILP_{cs} exploits the fact that, especially when the alphabet size is small, many substrings occur many times in both input strings. In the case of model ILP_{cb}, if a substring t appears, for example, 10 times in s_1 and 20 times in s_2, $10 \times 20 = 200$ common blocks are required just for this substring. In the following, let T denote the set of all (unique) strings that appear as substrings in both s_1 and s_2. For each $t \in T$, let $Q1_t$, respectively, $Q2_t$, denote the set of all positions at which t starts in input string s_1, respectively, s_2. For each combination of $t \in T$ and $k \in Q1_t$, model ILP_{cs} uses a binary variable $y^1_{t,k}$. In the same way, a binary variable $y^2_{t,k}$ is used for each $t \in T$ and $k \in Q2_t$. Based on these variables, model ILP_{cs} can be stated as follows:

$$\min \quad \sum_{t \in T} \sum_{k \in Q1_t} y^1_{t,k} \tag{4}$$

$$\text{s.t.} \quad \sum_{t \in T} \sum_{k \in Q1_t \text{ s.t. } k \leq j < k+|t|} y^1_{t,k} = 1 \quad \text{for } j = 1, \ldots, n \tag{5}$$

$$\sum_{t \in T} \sum_{k \in Q2_t \text{ s.t. } k \leq j < k+|t|} y^2_{t,k} = 1 \quad \text{for } j = 1, \ldots, n \tag{6}$$

$$\sum_{k \in Q1_t} y^1_{t,k} = \sum_{k \in Q2_t} y^2_{t,k} \quad \text{for } t \in T \tag{7}$$

$$y^1_{t,k} \in \{0, 1\} \quad \text{for } t \in T,\ k \in Q1_t$$

$$y^2_{t,k} \in \{0, 1\} \quad \text{for } t \in T,\ k \in Q2_t.$$

The objective function (4) corresponds to the number of selected substrings.[2] Equations (5) and (6) ensure that for each position $j = 1, \ldots, n$ of input strings s_1 and s_2, exactly one covering substring is selected. Furthermore, Equation (7) ensures that each string $t \in T$ is chosen the same number of times from s_1 and from s_2.

[2]In fact, $\sum_{t \in T} \sum_{k \in Q2_t} y^2_{t,k}$ can equally be used as the objective function.

The modification of ILP_{cs}, henceforth denoted by $\text{ILP}_{\text{cs}}^{\text{mod}}$, is obtained in a way which is analogous to the modification of model ILP_{cb}. In particular, let $T^{>1} \subset T$ be the subset of T containing all strings with length greater than 1. Model $\text{ILP}_{\text{cs}}^{\text{mod}}$ is then obtained from ILP_{cs} as follows:

1. All occurrences of T are replaced with $T^{>1}$.
2. A new objective function, to be minimized, is used: $\sum_{t \in T^{>1}} \sum_{k \in Q1_t} y^1_{t,k} + (n - \sum_{t \in T^{>1}} \sum_{k \in Q1_t} y^1_{t,k} |t|)$.
3. The equality signs in Equations (5) and (6) are replaced by a "$\leq$" symbol.

Finally, given an optimal solution to $\text{ILP}_{\text{cs}}^{\text{mod}}$, an optimal solution to ILP_{cs} can be derived in a way analogous to the procedure described in the context of models ILP_{cb} and $\text{ILP}_{\text{cb}}^{\text{mod}}$.

2.3. Results

In order to test the differences between the four ILP models, they were implemented for their resolution with the ILP solver IBM ILOG CPLEX v12.7. Hereby, ANSI C++ was used as programming language, and GCC 5.4.0 was used for compilation. Moreover, note that CPLEX was run in one-threaded mode, in order to be able to perform a fair comparison with the other algorithms proposed in this work (see Section 5). The experimental evaluation was performed on a cluster of PCs with Intel(R) Xeon(R) CPU 5670 CPUs of 12 nuclei of 2933 MHz and at least 40 GB of RAM. As the techniques proposed in this paper are thought for the application to large-scale instances, 10 instances were produced uniformly at random for each combination of $n \in \{2000, 4000, \ldots, 20,000\}$ and $|\Sigma| \in \{4, 20, 36, 52\}$. Thus, the benchmark set consists of 400 problem instances. Each problem instance was solved with each of the four models, with a computation time limit of 3600 CPU seconds, that is, one hour of computation time. The obtained results are provided in Table 1. Each table row provides the results for each of the four models, averaged over the 10 problem instances for a certain combination of n and $|\Sigma|$. For each model we provide four values: (a) the average solution quality (column with heading "avg."), (b) the time (in seconds) when the first integer feasible solution was found (column with heading "ti (s)"), (c) the time (in seconds) when the best integer feasible solution was found (column with heading "tb (s)"), and (d) the average optimality gap in percent (column with heading "Gap (%)"). All cases in which the search was aborted—due to exceeding the memory limit of 8 GB—before finding a first integer feasible solution are marked with "-s-." Moreover, those cases in which not even the data structures and the ILP model fit into the allowed memory of 8 GB are marked with "-m-." Finally, those few cases in which optimality was proved are marked with an asterisk. The best average result of each table row is provided in bold.

The following observations can be made:

- First of all, in both cases (ILP_{cb} and ILP_{cs}) CPLEX provides—when given the same computation time limit—generally better results with the modified models than with the original ones. This is with the exception of ILP_{cs} and $\text{ILP}_{\text{cs}}^{\text{mod}}$ in the context of instances based on alphabets with $|\Sigma| = 4$. Starting from string length $n = 1000$, CPLEX only finds the trivial solutions of the modified models (setting all decision variables to zero) within the allowed computation time.

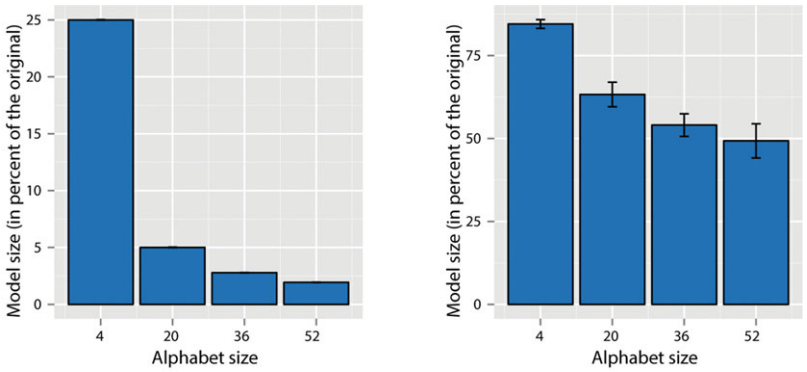
Table 1
Numerical results obtained from the four ILP models

		ILP_{cb}				$\text{ILP}_{\text{cb}}^{\text{mod}}$				ILP_{cs}				$\text{ILP}_{\text{cs}}^{\text{mod}}$			
$\lvert\Sigma\rvert$	n	avg.	ti (s)	tb (s)	Gap (%)	avg.	ti (s)	tb (s)	Gap (%)	avg.	ti (s)	tb (s)	Gap (%)	avg.	ti (s)	tb (s)	Gap (%)
4	2000	531.8	13.3	3593.9	23.9	539.8	4.5	3593.8	25.0	469.4	0.5	3526.2	13.7	**467.4**	0.4	3288.6	13.3
	4000	-s-	-s-	-s-	-s-	4000.0	17.7	17.7	44,497.8	**978.1**	1.1	3551.0	24.9	1008.2	0.9	3599.7	27.2
	6000	-s-	-s-	-s-	-s-	6000.0	285.7	285.7	66,713.2	**1394.9**	2.0	3579.4	25.2	2371.6	1.7	3581.0	39.6
	8000	-s-	-s-	-s-	-s-	-s-	-s-	-s-	-s-	**1805.8**	2.5	2.5	100.0	8000.0	2.3	2875.6	422.4
	10,000	-s-	-s-	-s-	-s-	-s-	-s-	-s-	-s-	**2192.0**	4.2	4.2	100.0	10,000.0	2.3	2.3	1871.6
	12,000	-m-	-m-	-m-	-m-	-s-	-s-	-s-	-s-	**2582.7**	4.9	4.9	100.0	12,000.0	3.3	3.3	1953.0
	14,000	-m-	-m-	-m-	-m-	-s-	-s-	-s-	-s-	**2952.7**	5.3	5.3	100.0	14,000.0	3.5	3.5	2018.8
	16,000	-m-	-m-	-m-	-m-	-s-	-s-	-s-	-s-	**3332.1**	7.1	7.1	100.0	16,000.0	4.9	4.9	2084.6
	18,000	-m-	-m-	-m-	-m-	-s-	-s-	-s-	-s-	**3713.3**	7.9	7.9	100.0	18,000.0	6.1	6.1	2136.6
	20,000	-m-	-m-	-m-	-m-	-m-	-m-	-m-	-m-	**4067.3**	10.3	10.3	100.0	20,000.0	7.9	7.9	2186.3
20	2000	1132.6	2.3	3597.0	14.1	987.9	0.2	3557.6	1.5	988.7	0.2	3599.7	1.5	**986.0**	0.1	3585.2	1.2
	4000	2069.9	8.7	2150.1	49.0	1794.9	0.5	3599.5	1.9	1803.2	0.3	3573.7	2.3	**1798.3**	0.2	3583.5	2.0
	6000	2962.6	22.8	22.8	100.0	2880.4	1.5	3595.6	13.1	2610.5	0.6	3098.3	4.1	**2575.1**	0.3	3327.7	2.8
	8000	3836.8	191.7	191.7	100.0	8000.0	3.4	3.4	2219.9	3593.9	0.8	3082.3	10.9	**3474.8**	0.6	2936.4	7.9
	10,000	-s-	-s-	-s-	-s-	10,000.0	5.0	5.0	2777.4	4414.8	1.0	3373.4	12.2	**4309.3**	0.6	3386.3	10.2
	12,000	-s-	-s-	-s-	-s-	12,000.0	8.7	8.7	3334.3	5351.5	1.7	3550.3	15.4	**5178.8**	0.6	3217.1	12.5
	14,000	-s-	-s-	-s-	-s-	14,000.0	13.5	13.5	3885.7	6228.3	1.8	3504.4	16.9	**6011.8**	0.8	3597.9	13.8
	16,000	-s-	-s-	-s-	-s-	16,000.0	12.2	12.2	4441.8	**7073.9**	2.8	2.8	100.0	16,000.0	1.4	1.4	303.8
	18,000	-s-	-s-	-s-	-s-	18,000.0	12.7	12.7	4995.3	**7852.6**	2.2	2.2	100.0	18,000.0	1.7	1.7	313.6
	20,000	-s-	-s-	-s-	-s-	20,000.0	14.5	14.5	5550.5	**8628.7**	2.2	2.2	100.0	20,000.0	1.5	1.5	321.8

Continued

Table 1
Continued

		ILP_{cb}				$\text{ILP}_{\text{cb}}^{\text{mod}}$				ILP_{cs}				$\text{ILP}_{\text{cs}}^{\text{mod}}$			
$\|\Sigma\|$	n	avg.	ti (s)	tb (s)	Gap (%)	avg.	ti (s)	tb (s)	Gap (%)	avg.	ti (s)	tb (s)	Gap (%)	avg.	ti (s)	tb (s)	Gap (%)
36	2000	1222.3	0.9	2805.6	0.1	**1221.8***	0.1	53.0	0.0	**1221.8***	0.1	107.3	0.0	**1221.8***	0.1	155.7	0.0
	4000	2490.2	4.3	3599.7	13.6	**2180.2**	0.2	3598.3	1.3	2185.1	0.3	3598.5	1.5	2181.0	0.2	3597.5	1.3
	6000	3567.6	11.1	11.1	100.0	3162.2	0.5	3268.2	3.2	3132.6	0.4	3409.7	2.3	**3119.1**	0.3	3599.2	1.9
	8000	4602.9	20.5	20.5	100.0	4183.7	0.7	3258.4	5.9	4148.9	0.9	2985.5	5.1	**4052.1**	0.4	2424.1	2.9
	10,000	5597.4	98.3	98.3	100.0	5112.4	1.3	3376.5	6.4	5062.4	1.2	2865.7	5.5	**4969.8**	0.6	2231.4	3.7
	12,000	6582.6	163.9	163.9	100.0	6039.2	1.6	3594.8	6.7	5943.9	0.9	2944.1	5.2	**5853.1**	0.6	3149.3	3.7
	14,000	-s-	-s-	-s-	-s-	14,000.0	2.3	2.3	1147.1	**6837.0**	1.0	2801.6	5.4	6839.7	0.6	3412.4	5.4
	16,000	-s-	-s-	-s-	-s-	16,000.0	3.2	3.2	1310.4	7752.8	1.3	3483.8	6.1	**7581.1**	0.8	3324.7	4.0
	18,000	-s-	-s-	-s-	-s-	18,000.0	4.0	4.0	1472.7	8815.3	2.5	3594.3	8.3	**8511.4**	0.7	3318.7	5.1
	20,000	-s-	-s-	-s-	-s-	20,000.0	5.1	5.2	1636.6	9598.4	2.5	3533.5	7.5	**9468.1**	1.0	3534.0	6.3
52	2000	**1412.8**	0.7	2.2	0.0	**1412.8***	0.1	0.1	0.0	**1412.8***	0.1	0.1	0.0	**1412.8***	0.1	0.1	0.0
	4000	2558.1	3.8	3222.7	3.3	**2476.5**	0.1	2026.2	<0.1	2476.7	0.2	2295.4	0.1	**2476.5**	0.1	2389.3	<0.1
	6000	3936.4	6.1	3608.4	12.4	**3479.4**	0.3	3588.7	0.9	3482.7	0.3	3600.0	1.0	3481.0	0.3	3598.5	0.9
	8000	5093.2	14.0	14.0	100.0	4468.9	0.4	3247.2	1.6	4554.6	0.7	3112.4	3.5	**4467.5**	0.3	3388.3	1.6
	10,000	6205.5	21.9	21.9	100.0	5468.5	0.6	3352.6	2.8	5593.4	1.0	3370.8	5.0	**5462.7**	0.5	2064.1	2.7
	12,000	7305.8	87.1	87.1	100.0	6436.4	0.8	3595.3	3.2	6580.2	1.4	3199.3	5.3	**6420.1**	0.5	2256.2	3.0
	14,000	8385.7	131.9	131.9	100.0	7625.2	1.6	3419.6	6.4	7553.9	1.0	2842.5	5.5	**7478.9**	0.8	3228.4	4.5
	16,000	-s-	-s-	-s-	-s-	8656.1	1.3	3413.1	7.0	8530.9	2.0	3466.2	5.6	**8494.4**	0.6	2952.7	5.2
	18,000	-s-	-s-	-s-	-s-	18,000.0	2.8	2.8	695.4	9515.1	2.1	3522.7	6.0	**9352.2**	0.6	3302.7	4.4
	20,000	-s-	-s-	-s-	-s-	20,000.0	3.4	3.4	771.7	10,826.5	2.6	3401.7	9.0	**10,424.2**	1.2	3523.8	5.6



(a) Relative sizes of model $\text{ILP}_{\text{cb}}^{\text{mod}}$. (b) Relative sizes of model $\text{ILP}_{\text{cs}}^{\text{mod}}$.

Fig. 2. Sizes of the modified models (in percent) with respect to the sizes of the original models (in terms of the number variables).

In contrast, when applied to ILP_{cs}, CPLEX finds better initial solutions in only slightly higher computation times. The same happens for instances with $|\Sigma| = 20$ and $n \geq 16{,}000$.

- Models ILP_{cs} and $\text{ILP}_{\text{cs}}^{\text{mod}}$ are the only ones that provide solutions for all problem instances. In contrast, ILP_{cb} can only provide solutions for 18 of 40 combinations of n and $|\Sigma|$. This is due to the extreme model size (see Table 2). For example, in the case of $n = 20{,}000$ and $|\Sigma| = 4$, model ILP_{cb} contains more than 133 million variables. CPLEX is clearly more efficient in solving the modified model $\text{ILP}_{\text{cb}}^{\text{mod}}$. However, even $\text{ILP}_{\text{cb}}^{\text{mod}}$ cannot be solved in 7 of 10 cases concerning the instances with $|\Sigma| = 4$.

Based on these results, we select model $\text{ILP}_{\text{cs}}^{\text{mod}}$ for its use within the CMSA and RVNS algorithms that are outlined in the following sections.

Finally, we also provide the information about the model size reduction obtained by our proposed modification of the ILP models in graphical terms. Figure 2 shows the size of the modified models in terms of the relative size (in percent) with respect to the sizes of the original models. The barplots provide this information in terms of averages over all instances of the same alphabet size. Standard deviations are also provided in these graphics. It can be seen that the model size reduction is greater for model ILP_{cb} than for model ILP_{cs}. Note that—in the case of $|\Sigma| = 52$, for example—the modified model ($\text{ILP}_{\text{cb}}^{\text{mod}}$) contains only about 2.5% of the variables of the original model.

3. Adaptation of the original CMSA algorithm

In the following, we first describe the redesigned CMSA algorithm, in contrast to the original one for solving the MCSP problem as described in Blum et al. (2016). Afterward, the main differences to the original CMSA version are outlined.

Our modified CMSA algorithm works on a complete set of *solution components* C, which consists of a component $c_{t,k}^1$ for each variable $y_{t,k}^1$ ($t \in T^{>1}$, $k \in Q1_t$) and a component $c_{t,k}^2$ for each variable $y_{t,k}^2$ ($t \in T^{>1}$, $k \in Q1_t$). The definition of variables $y_{t,k}^1$ and $y_{t,k}^2$ is found in Section 2.2. In short, a

Table 2
Absolute model sizes (in terms of the number of variables)

n	$\|\Sigma\| = 4$				$\|\Sigma\| = 20$				$\|\Sigma\| = 36$				$\|\Sigma\| = 52$			
	ILP_{cb}	ILP_{cb}^{mod}	ILP_{cs}	ILP_{cs}^{mod}	ILP_{cb}	ILP_{cb}^{mod}	ILP_{cs}	ILP_{cs}^{mod}	ILP_{cb}	ILP_{cb}^{mod}	ILP_{cs}	ILP_{cs}^{mod}	ILP_{cb}	ILP_{cb}^{mod}	ILP_{cs}	ILP_{cs}^{mod}
2000	1335.4	333.9	21.6	17.6	212.6	10.7	8.9	4.9	116.4	3.3	7.4	3.4	80.5	1.6	6.2	2.2
4000	5336.7	1334.5	47.2	39.2	846.0	42.4	19.4	11.4	461.0	12.9	16.4	8.4	317.3	6.2	14.5	6.5
6000	12,003.7	3001.4	74.4	62.4	1898.9	95.1	30.8	18.8	1033.5	28.8	25.4	13.4	711.0	13.8	23.3	11.3
8000	21,339.9	5334.7	102.3	86.3	3374.3	168.7	43.0	27.0	1835.9	51.2	34.6	18.6	1262.1	24.3	32.1	16.1
10,000	33,345.6	8338.8	131.3	111.3	5271.4	263.9	55.6	35.6	2867.1	79.9	44.0	24.0	1970.2	38.1	40.9	20.9
12,000	48,013.0	12,005.0	160.7	136.7	7588.5	380.1	68.5	44.5	4125.9	115.1	53.7	29.7	2834.8	54.7	49.8	25.8
14,000	65,350.5	16,341.4	190.4	162.4	10,327.0	516.7	81.6	53.6	5612.5	156.1	63.5	35.5	3856.9	74.5	58.6	30.6
16,000	85,351.9	21,340.4	220.9	188.9	13,488.4	675.2	94.9	62.9	7329.3	203.9	73.6	41.6	5036.2	97.2	67.4	35.4
18,000	108,016.6	27,005.1	251.4	215.4	17,067.6	854.2	108.3	72.3	9272.0	257.7	84.0	48.0	6370.2	122.7	76.4	40.4
20,000	133,350.8	33,341.2	282.5	242.5	21,069.6	1054.6	121.7	81.7	11,445.3	318.2	94.5	54.5	7862.3	151.4	85.4	45.4

Numbers are divided by 1000 and rounded to the first position after the comma.

setting of $y^1_{t,k} = 1$, respectively, $y^2_{t,k} = 1$, in model $\text{ILP}^{\text{mod}}_{\text{cs}}$ means that substring t starting at position k in input string s_1, respectively, input string s_2, is selected for the solution. Moreover, when the value of such a variable is 1, the corresponding solution component is added to the respective solution. This means that a solution S in the context of the CMSA algorithm is a subset of the complete set of solution components, that is, $S \subset C$. In the same way, a subinstance C' is a subset of the complete set of solution components. Furthermore, CMSA uses a so-called *age value* $a^j_{t,k} \geq 0$ for each solution component $c^j_{t,k} \in C$, $j \in \{1, 2\}$. These age values are used as a measure of the usefulness of solution components. Finally, the objective function value $f(S)$ of a solution S is (in accordance to the objective function of model $\text{ILP}^{\text{mod}}_{\text{cs}}$) calculated as $f(S) := |S|/2 + (n - \sum_{c^1_{t,k} \in S} |t|)$. Note that the term $|S|/2$ results from the fact that only solution components concerning the first input string are counted.

CMSA, whose pseudocode is provided in Algorithm 1, works roughly as follows. First, the subinstance C' is initialized with the components from an initially generated solution (see line 3 of Algorithm 1). Then, at each iteration, two main algorithmic are used: (a) the probabilistic construction of solutions (see line 7 of Algorithm 1) and (b) the solution of reduced problem instances (subinstances) of the original problem instances by means of an exact solver (see line 11 of Algorithm 1). In the case of the MCSP, subinstances are modeled as ILPs, and CPLEX is used as an exact solver. At each iteration, CMSA probabilistically generates a number of n_a solutions. The components found in these solutions are added to the current subinstance, which is then solved (if possible) to optimality by CPLEX. The components found in the solution provided by CPLEX receive an age value of 0, which means that they are regarded to be useful. The age value of all other components in the subinstance is incremented. The last action of each iteration consists in removing those components that have reached a maximum allowed age—denoted by age_{max}—from the current subinstance. The rationale behind this step is that components that never appear in the optimal solution of the current subinstance should be removed because they simply slow down CPLEX when solving the subinstance to optimality. In the following, the functions of CMSA are outlined in more detail.

Algorithm 1. Construct, merge, solve, and adapt (CMSA)

1: **input**: two related input strings s_1 and s_2. Values for the parameters
2: $a^j_{t,k} := 0 \; \forall \; c^j_{t,k} \in C$
3: $S_{\text{bsf}} :=$ GenerateInitialSolution()
4: $C' := S_{\text{bsf}}$
5: **while** termination conditions not satisfied **do**
6: **for** $i = 1, \ldots, n_a$ **do**
7: $S :=$ GenerateSolution()
8: $C' := C' \cup S$
9: **if** $f(S) < f(S_{\text{bsf}})$ **then** $S_{\text{bsf}} := S$ **end if**
10: **end for**
11: $S_{\text{ilp}} :=$ ApplyILPSolver(C')
12: **if** $f(S_{\text{ilp}}) < f(S_{\text{bsf}})$ **then** $S_{\text{bsf}} := S_{\text{ilp}}$ **end if**
13: $C' :=$ AdaptSubinstance(S_{ilp})
14: **end while**
15: **output**: S_{bsf}

Function GenerateInitialSolution(): As shown in Section 2.3, when using model ILP_{cs}, CPLEX is able to generate reasonably good integer feasible solutions in a relatively short amount of computation time. Therefore, function GenerateInitialSolution() (see line 3 of Algorithm 1) applies CPLEX to model ILP_{cs} and stops the run when the first integer solution is found. The solution components corresponding to those variables representing substrings of length greater than 1 are added to S_{bsf}. Moreover, the subinstance C' is initialized with the components from S_{bsf} (see line 4 of Algorithm 1).

Function GenerateSolution(): This function uses a modified variant of the greedy algorithm from (He, 2007) in a probabilistic way. This algorithm is based on the concept of common blocks. The original deterministic version works as follows. Given a valid partial solution $S \subset B$, let $N(S) \subset B$ denote the set of common blocks that can be added to S such that the result is again a valid (possibly still partial) solution. The algorithm starts with $S := \emptyset$. At each iteration, one of the blocks $b_i \in N(S)$ is selected such that

$$b_i := \text{argmax}\{|t_j| \mid b_j \in N(S)\}. \tag{8}$$

This block is then added to S, and the process is continued until S is a complete—in the sense of nonextensible—solution. In other words, the algorithm stops once $N(S)$ is empty. The modified version of this algorithm uses only the blocks from $B^{>1}$, that is, blocks representing substrings of length greater than 1. However, during preliminary experiments we noted that even this modified version runs into memory problems for instances with $n \geq 18{,}000$ and $|\Sigma| = 4$. Therefore, the algorithm is executed with $B^{>2}$ instead of $B^{>1}$, that is, only considering blocks representing substrings of length greater than 2. Once no more such block can be added to S, we only generate those blocks from $B^{=2}$—that is, the set of blocks representing substrings of length 2—that can feasibly be added to S, and proceed with the algorithm until no further block with a substring of length 2 can be added. Function GenerateSolution() (see line 7 of Algorithm 1) uses this modified algorithm in a probabilistic way. In particular, at each construction step, the longest feasible block according to Equation (8) is selected with a predefined probability $d_{\text{rate}} \geq 0$. Otherwise, a candidate list of the $l_{\text{size}} > 0$ best candidates is generated, and one of these candidates is selected uniformly at random. Both d_{rate} and l_{size} are important parameters of CMSA. Finally, a solution S is returned in terms of a set of solution components, instead of a set of common blocks. Note that each selected block $b_i = (t_i, k1_i, k2_i)$ is easily transformed into two solution components $c^1_{t_i,k1_i}$ and $c^2_{t_i,k2_i}$.

Function ApplyILPSolver(C'): This function (see line 11 of Algorithm 1) applies model $\text{ILP}^{\text{mod}}_{\text{cs}}$ to subinstance C', that is, instead of applying $\text{ILP}^{\text{mod}}_{\text{cs}}$ considering the complete set of variables, only those variables corresponding to the solution components in C' are considered. A fixed computation time limit $t_{\text{cplex}} > 0$ is given to CPLEX for this purpose. Note that t_{cplex} is an important parameter of CMSA.

Function AdaptSubinstance(S_{ilp}): In this function (see line 13 of Algorithm 1), first, the age of all components in C' is incremented by 1. Then, the age of all components that are in S_{ilp} is set

to 0. Moreover, all components from C' whose age has reached the maximum age limit $\text{age}_{\max}$ are removed from C'.

Note that the differences of this version of CMSA from the original version for the MCSP problem as described in Blum et al. (2016) are as follows. First, the original version does not use any initialization of C', that is, it starts with $C' := \emptyset$. Second, it uses the original version of the greedy algorithm by He (2007). Third, it uses model ILP_{cb} in line 11 for solving the current subinstance C' to optimality.

4. ILP-based reduced variable neighborhood search

RVNS (Mladenović and Hansen, 1997) is obtained from standard VNS by removing the local search phase. In other words, the algorithm explores the predefined neighborhoods randomly by means of the shaking procedure. As mentioned in Mladenović and Hansen (1997), RVNS is thought for the application to large-scale problem instances for which the local search step might be computationally too heavy. Successful applications of RVNS from the literature include the ones to uncapacitated multilevel lot-sizing problems (Xiao et al., 2011), biomedical literature extraction and clustering (Consoli and Stilianakis, 2017), and the robust dynamic maximal covering location problem (Mišković, 2017). Moreover, RVNS is sometimes used to rapidly produce reasonably good initial solutions (Kocatürk and Özpeynirci, 2014), or in combination with population-based techniques (Coelho et al., 2016).

Algorithm 2. Reduced variable neighborhood search (RVNS)

1: **input**: two related input strings s_1 and s_2. Neighborhood functions N_k, $k = 1, \ldots, k_{\max}$
2: $S_{\text{bsf}} :=$ GenerateInitialSolution()
3: $k := 1$
4: **while termination conditions not satisfied do**
5: $S' :=$ ChooseRandomNeighbor($N_k(S_{\text{bsf}})$)
6: **if** $f(S') < f(S_{\text{bsf}})$ **then**
7: $S_{\text{bsf}} := S'$, $k := 1$
8: **else**
9: $k \leftarrow k + 1$
10: **if** $k > k_{\max}$ **then** $k := 1$ **end if**
11: **end if**
12: **end while**
13: **output**: S_{bsf}

Just like CMSA, the proposed RVNS—whose pseudocode is provided in Algorithm 2—works on the complete set of solution components C defined in the previous section. Moreover, solutions are defined in the same way as in the case of CMSA, that is, a solution S is a subset of C and its objective function value $f(S)$ is defined as $|S|/2 + (n - \sum_{c^1_{t,k} \in S} |t|)$. In the following, the functions of Algorithm 2 are described in detail. First, the function for generating the initial solution (see

GenerateInitialSolution() in line 2 of Algorithm 2 is equal to the function in line 3 of Algorithm 1. Second, the most important function—ChooseRandomNeighbor($N_k(S_{\text{bsf}})$); see line 5 of Algorithm 2—uses a neighborhood function defined via a solution destruction rate. More specifically, given a destruction rate $0 < d < 1$, a random neighbor of the current solution S_{bsf} is obtained by randomly removing $\lfloor |S_{\text{bsf}}| \cdot d \rfloor$ solution components from S_{bsf} and subsequently completing the resulting partial solution S^p_{bsf} by applying model $\text{ILP}^{\text{mod}}_{\text{cs}}$ with the following additional set of constraints:

$$y^j_{t,k} = 1 \quad \forall\, c^j_{t,k} \in S^p_{\text{bsf}}, \quad j \in \{1, 2\}. \tag{9}$$

The partial destruction of solution S_{bsf} works as follows. First, S_{bsf} is copied into S^p_{bsf}. Then, while $|S^p_{\text{bsf}}| > |S_{\text{bsf}}| - \lfloor |S_{\text{bsf}}| \cdot d \rfloor$, a component—say $c^j_{t,k}$—is randomly chosen from S^p_{bsf} and successively removed. Moreover, in case $j = 1$ (respectively, $j = 2$), a corresponding component $c^2_{t,l} \in S^p_{\text{bsf}}$ (respectively, $c^1_{t,l} \in S^p_{\text{bsf}}$) is removed. Note that, due to the constraints of the problem as described in the context of ILP_{cs}, such a component must exist. Finally, a specific neighborhood $N_k()$ is defined via a minimum (respectively, maximum) destruction rate $d_{\min}$ (respectively, $d_{\max}$). The destruction rate d_k that is applied in the context of neighborhood function $N_k()$ is

$$d_k := d_{\min} + (k-1) \cdot \frac{(d_{\max} - d_{\min})}{k_{\max} - 1}. \tag{10}$$

Hereby, $k_{\max}$—that is, the number of neighborhoods—as well as $d_{\min}$ and $d_{\max}$ are important parameters of the algorithm. As in the case of CMSA, the application of CPLEX at each iteration of RVNS is limited with a computation time limit (in seconds) denoted by t_{cplex}.

5. Experimental evaluation

The following three algorithmic techniques are experimentally evaluated: (a) the greedy algorithm by He (2007), modified according to the same reasoning as outlined in Section 2 (denoted by GREEDY), (b) the modified current state-of-the-art algorithm CMSA from Section 3, and (c) the ILP-based RVNS (henceforth denoted by RVNS) from Section 4. All algorithms were implemented in ANSI C++ using GCC 5.4.0. In addition, the ILP models involved in the three techniques were solved with the ILP solver IBM ILOG CPLEX v12.7 in one-threaded mode. As in the case of the experimental evaluation of the ILP models in Section 2.3, 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 GB of RAM.

5.1. Parameter tuning

Both CMSA and RVNS include important parameters for which well-working values must be found. In order to avoid having to apply a tuning procedure for each combination of n (input string length) and $|\Sigma|$ (alpha size), we decided for the following procedure. We performed a full

Table 3
Parameter value settings for CMSA and RVNS as determined by a full factorial design

		CMSA					RVNS		
$\|\Sigma\|$	n	n_a	$\text{age}_{\max}$	d_{rate}	l_{size}	t_{cplex}	$(d_{\min}, d_{\max})$	$k_{\max}$	t_{cplex}
4	2000	3	5	0.1	10	10.0	(0.3, 0.5)	3	50.0
	10,000	3	5	0.5	10	10.0	(0.3, 0.5)	5	50.0
	20,000	3	5	0.5	5	100.0	(0.3, 0.7)	10	100.0
20	2000	10	5	0.1	10	50.0	(0.3, 0.7)	3	100.0
	10,000	3	1	0.1	10	50.0	(0.3, 0.5)	3	100.0
	20,000	3	1	0.5	10	100.0	(0.1, 0.5)	3	100.0
36	2000	3	1	0.1	10	50.0	(0.1, 0.7)	5	10.0
	10,000	3	1	0.1	5	50.0	(0.1, 0.7)	5	50.0
	20,000	3	1	0.5	5	100.0	(0.1, 0.5)	3	50.0
52	2000	3	1	0.1	10	50.0	(0.1, 0.7)	3	50.0
	10,000	3	1	0.1	10	50.0	(0.1, 0.7)	3	50.0
	20,000	3	1	0.9	5	100.0	(0.1, 0.5)	3	50.0

factorial design based on the parameters and values as described below for each of the two algorithms and instances for each combination of $n \in \{2000, 10,000, 20,000\}$ and $|\Sigma| \in \{4, 20, 36, 52\}$. For this purpose, two tuning instances were randomly generated for each of these combinations of n and Σ. The considered parameters and their respective values in the case of CMSA are as follows:

- $n_a \in \{3, 5, 10\}$;
- $\text{age}_{\max} \in \{1, 5, 10\}$;
- $d_{\text{rate}} \in \{0.1, 0.5, 0.9\}$;
- $l_{\text{size}} \in \{3, 5, 10\}$;
- $t_{\text{cplex}} \in \{10.0, 50.0, 100.0\}$.

Furthermore, the considered parameters and their respective values in the case of RVNS are as follows:

- $(d_{\min}, d_{\max}) \in \{(0.1, 0.5), (0.3, 0.7), (0.1, 0.7), (0.3, 0.5)\}$;
- $k_{\max} \in \{3, 5, 10\}$;
- $t_{\text{cplex}} \in \{10.0, 50.0, 100.0\}$.

The best parameter value settings determined by full factorial design are provided in Table 3. Note that for instances with $n \notin \{2000, 10,000, 20,000\}$ the parameter values are determined by linear interpolation. In the case of integer-valued parameters, the actual values are hereby determined by rounding.

5.2. Numerical results

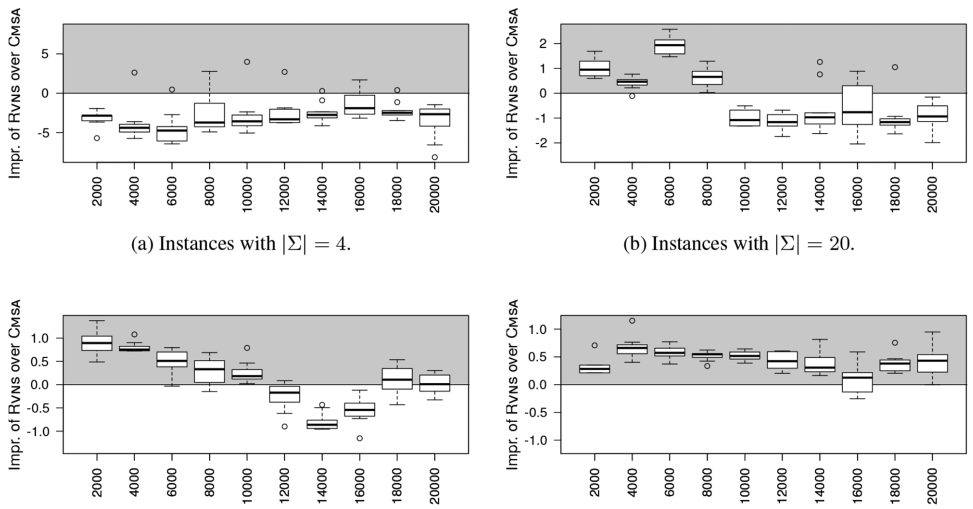
All three algorithms were applied exactly once—with a computation time limit of 3600 CPU seconds—to each of the 400 problem instances described in Section 2.3. The results are presented in Table 4 in terms of averages over the 10 instances for each combination of $|\Sigma|$ and n. In the case of GREEDY, CMSA, and RVNS, the columns with heading "avg" provide the average solution quality, whereas the columns with heading "t (s)" provide the average computation time at which the best solutions of each run were found. In addition, the third table column provides the best results obtained by any of the four ILP models from Section 2. The best result of each table row is provided in bold. The following can be observed:

- First, the results of the ILP models can only compete with (and slightly improve over) the results of CMSA and RVNS in the context of smaller problem instances based on alphabet sizes $|\Sigma| \in \{20, 36, 52\}$. With growing instance size we can note an increasing advantage of CMSA and RVNS over the ILP models. This effect becomes stronger with decreasing alphabet size.
- Even though being the fastest algorithm, GREEDY generally provides results inferior to the ones of the other techniques. This is with the exception of larger problem instances based on alphabet sizes $|\Sigma| \in \{36, 52\}$, where GREEDY is able to provide better results than the ILP models.
- Concerning the comparison of CMSA with RVNS, we can state that CMSA consistently outperforms RVNS for instances with $|\Sigma| = 4$, while the opposite is the case for instances with $|\Sigma| = 52$. For the instances with intermediate alphabet sizes, CMSA outperforms RVNS for the larger problem instances, whereas the opposite is the case for the smaller problem instances. The reason for this behavior might be explained as follows. In a previous study (see Lizárraga et al., 2017), it was shown—in the context of the multidimensional knapsack problem—that CMSA has advantages over large neighborhood search (LNS) for problem instances in which the size of a solution is rather small with respect to the total problem size, while the opposite was the case for problem instances with large solutions with respect to the total problem size. Hereby, the size of a solution refers to the number of variables with a nonzero value. The authors of Lizárraga et al. (2017) came up with the hypothesis that the above-mentioned behavior is due to the fact that CMSA generates, at each iteration, solutions potentially from all over the search space, while LNS is restricted to generate solutions locally around the currently best solution. Rather small solutions cause a disadvantage of LNS, due to the resulting difficulties to perform larger jumps in the search space. The same seems to hold for RVNS, which also generates solutions locally around the currently best solution. Moreover, note that—when alphabet sizes are small—solutions are rather small with respect to the total problem size, while the opposite is the case when alphabet sizes are rather large. In addition to the numerical results in Table 4, these findings are also shown in a graphical way in Fig. 3. The boxplots show for each combination of $|\Sigma|$ and n the differences of the objective function values obtained by CMSA with the objective function values obtained by RVNS (in percent). Accordingly, negative values indicate that CMSA has obtained better results than RVNS, and the other way around.

Finally, with the aim of detecting statistical differences (if any) between the studied techniques for subsets of the problem instances, first, all techniques were compared simultaneously using the Friedman test. As the test rejected the hypothesis that all the techniques perform equally,

Table 4
Numerical results obtained by GREEDY, CMSA, and RVNS

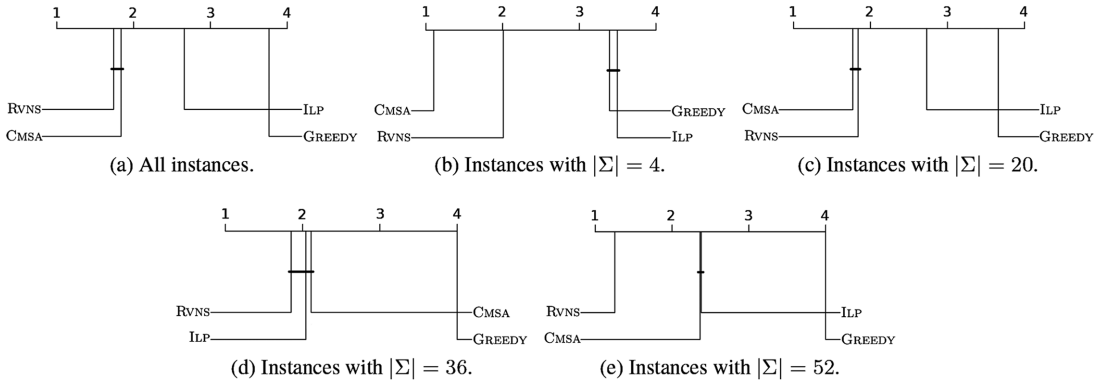
		ILP models	GREEDY		CMSA		RVNS	
$\|\Sigma\|$	n	Best result	avg.	t (s)	avg.	t (s)	avg.	t (s)
4	2000	467.4	527.4	0.3	**459.6**	2564.6	474.1	2600.3
	4000	978.1	968.4	1.4	**862.0**	3163.9	894.8	3331.3
	6000	1394.9	1379.7	3.0	**1237.2**	3272.4	1292.3	3227.4
	8000	1805.8	1791.5	5.9	**1634.0**	2933.5	1678.1	3314.1
	10,000	2192.0	2177.7	12.7	**1996.3**	2934.2	2052.6	3290.8
	12,000	2582.7	2562.8	14.8	**2361.6**	3297.7	2421.6	3118.2
	14,000	2952.7	2934.3	22.8	**2730.9**	3193.1	2797.0	2965.7
	16,000	3332.1	3300.2	27.5	**3105.7**	3327.4	3147.6	3048.3
	18,000	3713.3	-m-	-m-	**3434.7**	3231.4	3511.0	3052.0
	20,000	4067.3	-m-	-m-	**3776.5**	3293.5	3908.7	2459.2
20	2000	**986.0**	1111.9	<0.1	1002.0	2109.7	991.7	1654.8
	4000	**1798.3**	2031.6	<0.1	1824.6	2209.3	1817.0	3102.7
	6000	**2575.1**	2908.1	<0.1	2662.7	2898.8	2611.6	3517.7
	8000	3474.8	3751.4	0.2	3408.2	2364.9	**3386.4**	3513.2
	10,000	4309.3	4566.0	0.2	**4112.8**	3285.5	4153.6	3545.8
	12,000	5178.8	5372.0	0.3	**4838.9**	3434.3	4894.5	3575.7
	14,000	6011.8	6167.5	0.4	**5598.8**	3411.9	5637.5	3555.1
	16,000	7073.9	6939.7	0.5	**6326.9**	3022.6	6360.5	3550.9
	18,000	7852.6	7708.6	0.7	**7008.3**	3403.6	7076.4	3560.2
	20,000	8628.7	8468.9	0.9	**7715.0**	3498.0	7785.7	3599.4
36	2000	**1221.8**	1328.6	<0.1	1235.0	359.9	1223.8	1135.6
	4000	**2180.2**	2431.6	<0.1	2200.6	1207.6	2182.9	1202.9
	6000	**3119.1**	3478.9	<0.1	3137.8	2434.8	3122.5	2467.9
	8000	4052.1	4492.9	<0.1	4057.3	2628.6	**4045.1**	3074.9
	10,000	4969.8	5474.8	<0.1	4944.0	3274.9	**4931.3**	2982.9
	12,000	5853.1	6432.6	<0.1	**5822.5**	2914.6	5837.6	3139.5
	14,000	6837.0	7414.9	0.1	**6685.4**	3183.2	6738.4	3172.8
	16,000	7581.1	8331.0	0.2	**7542.8**	2871.1	7585.5	3328.3
	18,000	8511.4	9241.5	0.2	8387.0	3183.3	**8380.1**	3607.5
	20,000	9468.1	10,170.8	0.2	9226.4	3188.9	**9225.4**	3557.9
52	2000	**1412.8**	1469.6	<0.1	1417.4	1.6	**1412.8**	4.1
	4000	**2476.5**	2691.5	<0.1	2497.1	303.9	2480.5	653.5
	6000	**3479.4**	3841.4	<0.1	3500.3	1956.3	3479.7	2064.7
	8000	4467.5	4969.5	<0.1	4479.9	2974.9	**4456.9**	1622.8
	10,000	5462.7	6056.2	<0.1	5431.4	3118.1	**5403.4**	3000.9
	12,000	6420.1	7113.1	<0.1	6382.6	3490.6	**6355.7**	3289.7
	14,000	7478.9	8174.0	<0.1	7340.8	2433.2	**7312.2**	3231.1
	16,000	8494.4	9200.3	<0.1	8289.5	3050.4	**8279.6**	3517.3
	18,000	9352.2	10,215.6	0.1	9223.5	3135.4	**9188.2**	3364.5
	20,000	10,424.2	11,252.0	0.1	10,185.2	2673.6	**10,142.5**	3316.4



(a) Instances with $|\Sigma| = 4$. (b) Instances with $|\Sigma| = 20$.

(c) Instances with $|\Sigma| = 36$. (d) Instances with $|\Sigma| = 52$.

Fig. 3. Improvement of RVNS over CMSA (in percent).



(a) All instances. (b) Instances with $|\Sigma| = 4$. (c) Instances with $|\Sigma| = 20$.

(d) Instances with $|\Sigma| = 36$. (e) Instances with $|\Sigma| = 52$.

Fig. 4. Critical difference plots.

all pairwise comparisons were performed using the Nemenyi *post hoc* test (García and Herrera, 2008). The corresponding results are shown in Fig. 4 by means of so-called *critical difference* (CD) plots. In these plots, each considered technique is positioned on the horizontal axis according to its average ranking concerning the considered subset of instances. The CD is then computed (significance level of 0.05) and the performance of those techniques that have a difference lower than CD are regarded as statistically equivalent. This is indicated by bold horizontal bars joining the respective techniques. Note that all the tests and the plots have been generated using R's **scmamp** package (Calvo and Santafé, 2016), available at https://github.com/b0rxa/scmamp. Concerning the outcome, while there is no statistical difference between CMSA and RVNS when considering all 400 problem instances together (see Fig. 4a), there are significant differences between these two techniques for instances with $|\Sigma| = 4$ and with $|\Sigma| = 52$.

6. Conclusions and future work

In this paper, we have adapted the current state-of-the-art algorithm—CMSA—to be able to solve large-scale instances of the MCSP problem. This was achieved via a modification of the existing ILP models that allows application of general purpose ILP solvers such as CPLEX to much larger problem instances than has been possible so far. Moreover, on the basis of the modified ILP models, we introduced an ILP-based RVNS technique for the considered problem. By means of an exhaustive experimental study, it was shown that CMSA consistently outperforms RVNS in the context of problem instances based on small alphabets, while the opposite is the case for problem instances with large alphabets. For instances of intermediate alphabet sizes, CMSA generally performs stronger than RVNS for longer input strings, while the opposite is the case for instances with shorter input strings.

Concerning future work, we plan to combine the complementary strengths of CMSA and RVNS in order to obtain an algorithm that performs equally well for problems based on rather small alphabets and, at the same time, for problems based on rather large alphabets. One way to achieve this would be to add explicit mechanisms for allowing RVNS to perform larger jumps in the search space. Moreover, we plan to study the application of algorithms similar to CMSA and RVNS to related problems such as sorting by reversals, finding the edit distance when moves are allowed, and block edit problems, just to name a few.

References

Blum, C., 2018. ILP-based reduced variable neighborhood search for large-scale minimum common string partition. *Electronic Notes in Discrete Mathematics* 66, 15–22.

Blum, C., Festa, P., 2016. *Metaheuristics for String Problems in Bio-informatics, Computer Engineering Series—Metaheuristics Set*, Vol. 6. John Wiley & Sons, Hoboken, NJ.

Blum, C., Lozano, J.A., Pinacho Davidson, P., 2014. Iterative probabilistic tree search for the minimum common string partition problem. In Blesa, M.J., Blum, C., Voss, S. (eds), Proceedings of HM 20104—9th International Workshop on Hybrid Metaheuristics, Springer, Berlin, pp. 154–154.

Blum, C., Lozano, J.A., Pinacho Davidson, P., 2015. Mathematical programming strategies for solving the minimum common string partition problem. *European Journal of Operational Research* 242, 3, 769–777.

Blum, C., Pinacho, P., López-Ibáñez, M., Lozano, J.A., 2016. Construct, merge, solve and adapt: a new general algorithm for combinatorial optimization. *Computers & Operations Research* 68, 75–88.

Blum, C., Raidl, G.R., 2016. Computational performance evaluation of two integer linear programming models for the minimum common string partition problem. *Optimization Letters* 10, 1, 189–205.

Calvo, B., Santafé, G., 2016. scmamp: statistical comparison of multiple algorithms in multiple problems. *The R Journal* 8, 1, 248–256.

Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T., 2005. Computing the assignment of orthologous genes via genome rearrangement. Proceedings of the Asia Pacific Bioinformatics Conference 2005, Singapore, pp. 363–378.

Chrobak, M., Kolman, P., Sgall, J., 2004. The greedy algorithm for the minimum common string partition problem. In Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds), Proceedings of APPROX 2004—7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, Lecture Notes in Computer Science, Vol. 3122. Springer, Berlin, pp. 84–95.

Coelho, V.N., Coelho, I.M., Souza, M.J.F., Oliveira, T.A., Cota, L.P., Haddad, M.N., Mladenovic, N., Silva, R.C.P., Guimarães, F.G., 2016. Hybrid self-adaptive evolution strategies guided by neighborhood structures for combinatorial optimization problems. *Evolutionary Computation* 24, 4, 637–666.

Consoli, S., Stilianakis, N.I., 2017. A quartet method based on variable neighborhood search for biomedical literature extraction and clustering. *International Transactions in Operational Research* 24, 3, 537–558.

Cormode, G., Muthukrishnan, S., 2007. The string edit distance matching problem with moves. *ACM Transactions on Algorithms* 3, 2, 1–19.

Ferdous, S.M., Sohel Rahman, M., 2013. Solving the minimum common string partition problem with the help of ants. In Tan, Y., Shi, Y., Mo, H. (eds), Proceedings of ICSI 2013—4th International Conference on Advances in Swarm Intelligence, *Lecture Notes in Computer Science*, Vol. 7928. Springer, Berlin, pp. 306–313.

Ferdous, S.M., Sohel Rahman, M., 2015. An integer programming formulation of the minimum common string partition problem. *PloS One* 10, 7, e0130266.

Ferdous, S.M., Sohel Rahman, M., 2017. Solving the minimum common string partition problem with the help of ants. *Mathematics in Computer Science* 11, 2, 233–249.

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

Goldstein, A., Kolman, P., Zheng, J., 2005. Minimum common string partition problem: hardness and approximations. In Fleischer, R., Trippen, G. (eds), Proceedings of ISAAC 2004—15th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 3341. Springer, Berlin, pp. 484–495.

Goldstein, I., Lewenstein, M., 2011. Quick greedy computation for minimum common string partitions. In Giancarlo, R., Manzini, G. (eds), Proceedings of CPM 2011—22nd Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, Vol. 6661. Springer, Berlin, pp. 273–284.

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

He, D., 2007. A novel greedy algorithm for the minimum common string partition problem. In Mandoiu, I., Zelikovsky, A. (eds), Proceedings of ISBRA 2007—Third International Symposium on Bioinformatics Research and Applications, Lecture Notes in Computer Science, Vol. 4463. Springer, Berlin, pp. 441–452.

Jiang, H., Zhu, B., Zhu, D., Zhu, H., 2012. Minimum common string partition revisited. *Journal of Combinatorial Optimization* 23, 4, 519–527.

Kocatürk, F., Özpeynirci, Ö., 2014. Variable neighborhood search for the pharmacy duty scheduling problem. *Computers & Operations Research* 51, 218–226.

Kolman, P., 2005. Approximating reversal distance for strings with bounded number of duplicates. In Jedrzejowicz, J., Szepietowski, A. (eds), Proceedings of MFCS 2005—30th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 3618. Springer, Berlin, pp. 580–590.

Kolman, P., Waleń, T., 2007. Reversal distance for strings with duplicates: linear time approximation using hitting set. In Erlebach, T., Kaklamanis, C. (eds), Proceedings of WAOA 2007—4th International Workshop on Approximation and Online Algorithms, Lecture Notes in Computer Science, Vol. 4368. Springer, Berlin, pp. 279–289.

Lizárraga, E., Blesa, M.J., Blum, C., 2017. Construct, merge, solve and adapt versus large neighborhood search for solving the multi-dimensional knapsack problem: which one works better when? In Hu, B., López-Ibáñez, M. (eds), Proceedings of EvoCOP 2017—17th European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, Berlin, pp. 60–74.

Manning, C.D., Raghavan, P., Schütze, H., 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge.

Mišković, S., 2017. A VNS-LP algorithm for the robust dynamic maximal covering location problem. *OR Spectrum* 39, 4, 1011–1033.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. *Computers & Operations Research* 24, 11, 1097–1100.

Shapira, D., Storer, J.A., 2002. Edit distance with move operations. In Apostolico, A., Takeda, M. (eds), Proceedings of CPM 2002—13th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, Vol. 2373. Springer, Berlin, pp. 85–98.

Xiao, Y., Kaku, I., Zhao, Q., Zhang, R., 2011. A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems. *European Journal of Operational Research* 214, 2, 223–231.