A Comparative Analysis of Two Matheuristics by Means of Merged Local Optima Networks

Christian Blum[1,*] and Gabriela Ochoa[2]

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

[2]Computing Science and Mathematics
University of Stirling, Stirling FK9 4LA, UK
gabriela.ochoa@cs.stir.ac.uk

Abstract

We present a comparative analysis of two hybrid algorithms for solving combinatorial optimisation problems. The first one is a specific variant of an established family of techniques known as large neighbourhood search (LNS). The second one is a much more recent algorithm known as construct, merge, solve & adapt (CMSA). Both approaches generate, in different ways, reduced sub-instances of the tackled problem instance at each iteration. The experimental analysis is conducted on two NP-hard combinatorial subset selection problems: the multidimensional knapsack problem and minimum common string partition. The results support the intuition that CMSA has advantages over the LNS variant in the context of problems for which solutions contain rather few items. Moreover, they show that the opposite may be the case for problems in which solutions contain rather many items. The analysis is supported by a new way of visualising the trajectories of the compared algorithms in terms of merged monotonic local optima networks.

1 Introduction

Hybrid metaheuristics for combinatorial optimisation [47, 8] have become increasingly popular in recent years due to their ability to combine the strengths of different ways of solving optimisation problems within a single algorithm. This holds, in particular, for algorithms that combine heuristic search with exact techniques. These algorithms are often labelled as matheuristics [9]. Algorithms related to large neighbourhood search (LNS) [41] and to very large-scale neighbourhood search [1] are probably among the most well-known techniques from this field. In this context, note that it is often possible to explain these algorithms using different terminology. Many LNS-based algorithms can be described, for example, in terms of ejection chain approaches [22]. In principle there are many ways of generating so-called *large neighbourhoods* for a given problem. However, many LNS approaches are based on the principle of *ruin-and-recreate* [46], also sometimes found as *destroy-and-recreate* or *destroy-and-rebuild*. At each iteration, first the incumbent solution is partially destroyed. Then, either an exact technique or any other appropriate technique is applied for finding—among all solutions that include the produced partial solution—a solution that improves the incumbent solution. Generally, a time limit is imposed on this step. Many examples of this type of LNS can be found in the literature, including [16, 45, 19], just to name a few. The large neighbourhoods generated in this context are known as destruction-based large neighbourhoods. Note that, for the ease of notation, in the remainder of this paper the term LNS will refer to LNS variants making use of destruction-based large neighbourhoods. However, keep in mind that there are alternative ways of defining large neighbourhoods that are used in algorithms such as local branching [20], the corridor method [10], and POPMUSIC [32]. In the latter approach (POPMUSIC), a large neighbourhood is—at each iteration—generated as follows. The incumbent solution is first split into parts. A so-called seed-part is then chosen and extended by adding other parts that are close to the seed-part in order to form a sub-problem. This step depends on some distance measure between solution parts. Finally, the generated sub-problem is solved by an approximate or an exact solution approach. This process is repeated until the incumbent solution does not contain a sub-problem that can be improved.

One of the most recent, generally applicable, hybrid metaheuristics that was proposed in the literature is construct, merge, solve and adapt (CMSA) [7]. The principal idea of CMSA is very similar to the one found in many LNS variants, namely, the iterative application of an exact technique to reduced problem instances, that is, sub-instances of the original problem instance.[1] The way, however, in which the sub-instances are produced in CMSA is different to how this is generally done in LNS. CMSA does not force the existence of a certain partial solution when using the exact technique. It rather reduces the number of options for building a feasible solution, and asks the exact technique for the best solution that can be built from the reduced set of options. Related ideas can be found in the following works. Applegate et al. [3] and Cook and Seymour [13] tackle the classic travelling salesman problem (TSP) in the following way. In a first phase, they generate a set of high-quality TSP solutions using a metaheuristic. These solutions are then merged, resulting in a reduced problem instance, which is then solved to optimality by an exact solver. Another example concerns the so-called *generate-and-solve* (GS) framework that was originally presented in [36]. The latest application of this framework can be found in [17]. Another prominent example of related work concerns *kernel search* [2], which is a heuristic framework based on the identification of a restricted set of promising solution components (called the kernel) and on the exact solution of sub-instances by ILP solvers. Applications include [24, 44]. Finally, ideas related to CMSA can also be found in research on evolutionary algorithms, where *solution merging* refers to the idea of exploring the union of two or more solutions by means of a specialised (for example, exact) technique. One of the latest applications can be found in [5].

Summarising, the common idea of both LNS and CMSA is to identify substantially reduced sub-instances that contain high-quality solutions to the original problem instance. These sub-instances are required to be small enough for the application, for example, of an exact technique within a reasonable time limit. In other words, both algorithms make use of techniques for reducing the search space of the considered problem instances.

[1]Note that the terms *reduced problem instance* and *sub-instance* refer to the same concept and will be used interchangeably.

1.1 Contribution of this Paper and Prior Work

This study is conducted in the context of so-called *subset selection problems*, which can be formally defined as follows:

1. C is a finite set of n items.
2. $F : 2^C \mapsto \{\text{TRUE}, \text{FALSE}\}$ is a function that decides for each subset of C if the subset corresponds to a feasible solution. Let $X \subseteq 2^C$ be the set of all feasible solutions.
3. $f : X \mapsto \mathbb{R}$ is an objective function that assigns a value to each feasible solution.

The goal is to find a feasible solution that minimises (or maximises) the objective function. Many well-known combinatorial optimisation problems can be expressed in terms of a subset selection problem. Consider, for example, the symmetric TSP. The set E of edges of the complete TSP graph $G = (V, E)$ corresponds to the set C of items. Function F evaluates a subset $S \subseteq E$ to be a feasible solution if and only if the edges from S form a Hamiltonian cycle in G. And finally, the objective function value $f(S)$ of a feasible solution S is determined by summing the distances of all edges in S. The goal in the case of the TSP is to minimise f.

In prior work [34] we started to study LNS and CMSA in a comparative way, focusing on the consequences of producing reduced problem instances at each iteration in different ways. Remember that—in the context of this paper—the term LNS refers to LNS variants in which a large neighborhood is generated, at each iteration, by the partial destruction of the incumbent solution. In particular, we started to study the following intuition—in the context of the multi-dimensional knapsack problem (MDKP) [28], which is a subset selection problem—in a systematic way.

> **Intuition:** Due to the different ways in which LNS and CMSA generate sub-instances, CMSA generally works better than LNS in the context of problem instances for which solutions are rather small, and the opposite is the case in the context of problem instances for which solutions are rather large.

In the context of the MDKP the *size of a solution* is measured as the number of items in the knapsack. In particular, varying the tightness of the capacity constraints the MDKP allows to generate problem instances over the whole range (from instances with small to instances with large solutions).

In this paper, we make the following contributions. First of all, we repeat the study concerning the MDKP. This is because the computational resources and the time that were available for the initial study from [34] were limited. Therefore, the parameter domains chosen for tuning were quite reduced. Second, we extend our comparative study of CMSA with the chosen LNS variant to another subset selection problem: minimum common string partition (MCSP) [23]. This was done with the aim of gathering evidence that would support the above-mentioned intuition more generally for subset selection problems. Note that the class of subset selection problems is a prominent class of combinatorial optimisation problems. Apart from the TSP and the MDKP, many other well known problems such as various knapsack problems, feature selection, as well as many graph-based problems such as graph colouring, minimum dominating set problems and maximum independent set problems can be modelled as subset selection problems [42].

Our above-mentioned intuition about the relative performance of CMSA and LNS is based on the consideration that, for reaching a high-quality solution, LNS needs to find a trajectory of overlapping solutions from the starting solution to the high-quality solution. The smaller the solutions are—that is, the fewer items they contain—the more difficult it should be for LNS to find such a trajectory. A theoretical validation of our intuition seems, a priori, rather difficult to achieve. Therefore, empirical evidence is gathered by studying the two subset selection problems (MDKP and MCSP) mentioned above. As will be outlined later, for both problems it is possible to generate problem instances over the whole range between instances with small solutions and instances with large solutions.

In addition to these algorithmic contributions, we study ways of comparing algorithms based on visualising their search trajectories together for the same problem instance by means of merged local optimal networks. Local Optima Networks (LONs) [38, 48] are a coarse-grained model of fitness landscapes inspired by work on energy surfaces in theoretical chemistry [18]. The idea is to compress the search space into a smaller mathematical object: a graph, where vertices are the local optima in the underlying landscape and edges are the possible search transitions among optima with a given search operator. LONs capture the number, distribution and connectivity pattern of local optima in the underlying landscape. The original LON models [15, 38, 48] were extracted by fully enumerating the studied search spaces and considering all possible transitions among optima, that is, deteriorating, improving and equal-fitness transitions. More recently, and in order to study the *funnel*[2] structure of combinatorial landscapes [37, 39, 40], a model called Monotonic LON (MLON) was introduced. In the MLON model, only the non-deteriorating transitions (ie. improving and equal) among optima are recorded. As the deteriorating edges are removed, this model produces less densely connected networks that are easier to analyse and visualise. MLONs have been studied for both fully enumerated [40] and sampled search spaces [37, 39]. We therefore consider sampled MLONs in our analysis of hybrid metaheuristics. Once a MLON has been constructed, a variety of metrics and visualisation tools can be applied to enlighten our understanding of its structure. MLONs have been applied to study the search space of well-known combinatorial optimisation problems in the context of standard local search neighbourhoods [37, 40, 39]. Our contributions in this article are twofold. First, to apply the model to large neighbourhoods within hybrid metaheuristics. Note that, in this setting, the nodes of the networks are the result of applying an exact solver to the sub-instance under consideration. The second contribution is to merge the the models representing the trajectories of two different algorithms, in order to visually contrast their search dynamics.

Finally, note that it is not the aim of this study to develop new state-of-the-art algorithms for the MDKP and MCSP. The aim is rather to draw general conclusions from the comparative study of LNS and CMSA in order to be able to decide which algorithm to use when faced with a new problem.

[2]A funnel refers to a grouping of local optima, forming a coarse-level gradient towards a low cost solution at the end. When sub-optimal funnels exist, search can get trapped and fail to reach the global optimum despite a large computing time.

1.2 Outline of the Paper

This paper is organised as follows. Section 2 first describes both LNS and CMSA in a general, problem-independent way. Then, in following subsections, the application of both algorithms to the MDKP and the MCSP are outlined. The empirical study is presented in Section 4, while the conclusions and possible lines for future work are provided in Section 5.

2 General Algorithm Descriptions

Both LNS and CMSA are general algorithms for solving combinatorial optimisation problems, and can therefore be described in a problem-independent manner. However, in order to gain readability, the description given in the following assumes (1) that the algorithm is applied to a subset selection problem and (2) that the exact technique used to solve sub-instances is a general-purpose integer linear programming (ILP) solver. A general instance I of any subset selection problem consists of a set C of items, a subset $\mathcal{S} \subseteq 2^C$ which contains all subsets of C that are feasible solutions to the problem instance, and an objective function $f : \mathcal{S} \mapsto \mathbb{R}$ that is to be minimised. In the case of the well-known travelling salesman problem, for example, C consists of all edges of an input graph G. A subset $S \subset C$ is a feasible solution—that is, $S \in \mathcal{S}$—if and only if the items/edges in S form a Hamiltonian path in the input graph G. Finally, in the context of CMSA any sub-instance—denoted by C'—is also a subset of C. Solutions to C' may only be formed by items from C'.

2.1 Destruction-Based Large Neighbourhood Search

The pseudo-code of a general destruction-based LNS using an ILP solver for solving the corresponding sub-instance at each iteration is provided in Algorithm 1. To start with, the initial solution S_{cur} is generated in function $\mathsf{GenerateInitialSolution}(I)$ (see line 2). A greedy heuristic is used for this purpose in most cases. At each iteration, the following actions are performed. First, the current incumbent solution S_{cur} is partially destroyed; see function $\mathsf{DestroyPartially}(S_{\text{cur}}, D_r, dest_{\text{type}})$ in line 6. The degree of destruction depends hereby on a parameter D_r called *destruction rate*, while the type of destruction depends on parameter $dest_{\text{type}}$. There are potentially different ways for the partial destruction of a solution. The least sophisticated way, which is probably applied in most cases, consists in doing this randomly. However, one might also think about a heuristically guided way of partially destructing solutions. In any case, the resulting partial solution S_{partial} is passed to the ILP solver; see function $\mathsf{Reconstruct}(S_{\text{partial}}, t_{\max})$ in line 7. This function receives—apart from S_{partial}—a time limit $t_{\max}$ as input. For the application of this function, the ILP solver is forced to exclusively consider solutions that contain S_{partial}. In other words, the corresponding sub-instance consists of all valid solutions to I that contain S_{partial}. The function provides S'_{opt}, the best valid solution found within $t_{\max}$ CPU seconds, as output. As a computation time limit is used, note that this solution is not necessarily an optimal solution to the sub-instance. Finally, the better solution between S'_{opt} and S_{cur} is chosen to be the incumbent solution for the next iteration. This might seem restrictive, because other—more probabilistic—ways of selecting between S'_{opt} and S_{cur} are possible. However, the LNS algorithm studied in this work is—in turn—equipped with a variable destruction rate $D^l \leq D_r \leq D^u$, which is handled in the style of the neighbourhood size in variable neighbourhood search (VNS) algorithms [26]. More in detail, if S'_{opt} is better than S_{cur}, the

Algorithm 1 Destruction-Based Large Neighborhood Search (LNS)

```
1: input: problem instance $I$, values for parameters $D^l$, $D^u$, $D^{\text{inc}}$, $dest_{\text{type}}$, and $t_{\max}$
2: $S_{\text{cur}} := \mathsf{GenerateInitialSolution}(I)$
3: $S_{\text{bsf}} := S_{\text{cur}}$
4: $D_r := D^l$
5: while CPU time limit not reached do
6:    $S_{\text{partial}} := \mathsf{DestroyPartially}(S_{\text{cur}}, D_r, dest_{\text{type}})$
7:    $S'_{\text{opt}} := \mathsf{Reconstruct}(S_{\text{partial}}, t_{\max})$
8:    if $S'_{\text{opt}}$ is better than $S_{\text{bsf}}$ then $S_{\text{bsf}} := S'_{\text{opt}}$
9:    if $S'_{\text{opt}}$ is better than $S_{\text{cur}}$ then
10:      $S_{\text{cur}} := S'_{\text{opt}}$
11:      $D_r := D^l$
12:   else
13:      $D_r := D_r + D^{\text{inc}}$
14:      if $D_r > D^u$ then $D_r := D^l$
15:   end if
16: end while
17: return $S_{\text{bsf}}$
```

value of D_r is returned to the lower bound D^l. Otherwise, the value of D_r is incremented by D^{inc}, which is also a parameter of the algorithm. If, after this update, the value of D_r is greater than the upper bound D^u, it is equally returned to the lower bound D^l. A proper choice of values for the lower bound D^l and for the upper bound D^u enables the algorithm to escape from local minima.

2.2 Construct, Merge, Solve and Adapt

Algorithm 2 provides the pseudo-code of an ILP-based CMSA for subset selection problems. Before starting with the first iteration, the best-so-far solution S_{bsf} is set to NULL, indicating that this solution is not yet initialised. Furthermore, the reduced problem instance $C' \subseteq C$ is initialised to the empty set. Each item $c \in C$ has a so-called *age value* denoted by $age[c]$ indicating for each item for how many iterations it forms already part of the reduced sub-instance C'. These age values are all initialised to zero. Then, at each iteration of CMSA the following actions are performed. First, the reduced problem instance C' is augmented in the following way (see lines 5 to 11). Function $\mathsf{ProbabilisticSolutionGeneration}(C)$ is used to generate n_{a} solutions in a probabilistic way. Moreover, the items found in the constructed solutions are added to C' and their age value is set to zero. Subsequently, the ILP solver is applied in function $\mathsf{SolveSubinstance}(C', t_{\max})$, with a time limit of $t_{\max}$ CPU seconds. The best (and hopefully optimal) solution found within the allowed time (S'_{opt}) is provided as output of the function. If S'_{opt} is better than the current best-so-far solution S_{bsf}, solution S'_{opt} is adopted as the new best-so-far solution. Next, the reduced sub-instance C' is adapted in function $\mathsf{Adapt}(C', S'_{\text{opt}}, age_{\max})$ depending on S'_{opt} and depending on the age values of the items; see line 14. More specifically, the age value of each item in $C' \setminus S'_{\text{opt}}$ is incremented by 1, while the age value of each item in $S'_{\text{opt}} \subseteq C'$ is re-initialised to zero. After that, those items from C' with an age value greater than $age_{\max}$—which is also a parameter of the algorithm—are deleted from C'. This has the effect that items that never appear in solutions

Algorithm 2 Construct, Merge, Solve and Adapt (CMSA)

```
1: **input:** problem instance $I$, values for parameters $n_a$, $age_{\max}$, and $t_{\max}$
2: $S_{bsf}$ := NULL; $C'$ := $\emptyset$
3: $age[c] := 0$ for all $c \in C$
4: while CPU time limit not reached do
5:    for $i := 1, \ldots, n_a$ do
6:       $S$ := ProbabilisticSolutionGeneration($C$)
7:       for all $c \in S$ and $c \notin C'$ do
8:          $age[c] := 0$
9:          $C' := C' \cup \{c\}$
10:      end for
11:   end for
12:   $S'_{opt}$ := SolveSubinstance($C'$, $t_{\max}$)
13:   if $S_{bsf}$ = NULL or $S'_{opt}$ is better than $S_{bsf}$ then $S_{bsf} := S'_{opt}$
14:   Adapt($C'$, $S'_{opt}$, $age_{\max}$)
15: end while
16: return $S_{bsf}$
```

returned by the ILP solver do not unnecessarily bloat C' in subsequent iterations. On the other side, components which appear in the solutions returned by the ILP solver should, of course, be maintained in C'.

2.3 Differences in Generating Reduced Sub-Instances

Although CMSA and (destruction-based) LNS are based on the same principal idea, the way in which this idea is implemented is quite different. In the case of LNS, the reduced sub-instance at each iteration is generated by, first, partially destroying the incumbent solution, which results in a partial solution S^p, and then defining the sub-instance such that any feasible solution must: (1) be a feasible solution to the original problem and (2) contain S^p. On the other side, CMSA produces a reduced sub-instance as follows. A set of solutions is constructed in a probabilistic way at each iteration. The items found in these solutions are added to the so-called sub-instance C', which is initially empty. The sub-instance is then defined such that any feasible solution must contain, exclusively, items from C'. Note that, since $C' \subseteq C$, any feasible solution to the sub-instance is, of course, also a feasible solution to the original problem instance.

3 Application to Two Different Subset Selection Problems

The comparative study of CMSA with the chosen LNS variant is conducted on two different subset selection problems. The first one is the well-known multi-dimensional knapsack problem (MDKP), and the second one is minimum common string partition (MCSP). These two problems were chosen due to the different characteristics of their ILP models. While the number of variables and constraints in the case of the MDKP is linear in the number of items, the number of variables in the case of the MCSP is exponential in the parameters of a problem instance. In the following, the application of CMSA and the chosen LNS variant to both problems is described.

Example 1: Small MDKP instance

Instance data: $n = 10$ **(10 items), and** $m = 2$ **(two resources)**

$$\begin{pmatrix} p_1 = 49 & r_{1,1} = 10 & r_{1,2} = 24 \\ p_2 = 48 & r_{2,1} = 34 & r_{2,2} = 9 \\ p_3 = 95 & r_{3,1} = 30 & r_{3,2} = 67 \\ p_4 = 73 & r_{4,1} = 4 & r_{4,2} = 92 \\ p_5 = 35 & r_{5,1} = 18 & r_{5,2} = 27 \\ p_6 = 60 & r_{6,1} = 60 & r_{6,2} = 16 \\ p_7 = 111 & r_{7,1} = 68 & r_{7,2} = 92 \\ p_8 = 93 & r_{8,1} = 66 & r_{8,2} = 78 \\ p_9 = 51 & r_{9,1} = 40 & r_{9,2} = 40 \\ p_{10} = 53 & r_{10,1} = 65 & r_{10,2} = 36 \end{pmatrix}$$

Resource capacities: $\text{cap}_1 = 79$, $\text{cap}_2 = 96$

The optimal solution is $S^* = \{c_1, c_3\}$ with objective function value $f(S^*) = 49 + 95 = 144$.

3.1 Application to the MDKP

The MDKP is a well studied NP-hard combinatorial optimisation problem belonging to the class of subset selection problems. Moreover, it is a popular test case for new algorithmic proposals (see, for example, [12, 33, 29]). The problem is technically defined as follows. Given is a set $C=\{c_1, \ldots, c_n\}$ of n items, and a number of m different resources. Hereby, the k-th resource ($k = 1, \ldots, m$) is available in a certain quantity (*capacity*) $\text{cap}_k > 0$, and each item $c_i \in C$ requires from the k-th resource ($k = 1, \ldots, m$) a given amount $r_{i,k} \geq 0$ (*resource consumption*). Moreover, each item $c_i \in C$ has associated a profit $p_i > 0$.

A solution (subset) $S \subseteq C$ is feasible if—for each resource $k = 1, \ldots, m$—the total consumption over all selected items ($\sum_{c_i \in S} r_{i,k}$) does not exceed the resource capacity cap_k. Moreover, a valid solution S is called non-extensible, if no $c_i \in C \setminus S$ can be added to S without destroying its property of being a valid solution. The aim is to find a feasible solution S of maximum total profit ($\sum_{c_i \in S} p_i$). The standard ILP formulation of the MDKP is as follows:

$$\text{maximize} \sum_{c_i \in C} p_i \cdot x_i \tag{1}$$

subject to:

$$\sum_{c_i \in C} r_{i,k} \cdot x_i \leq \text{cap}_k \quad \forall k = 1, \ldots, m \tag{2}$$

$$x_i \in \{0, 1\} \qquad \forall c_i \in C \tag{3}$$

Note that this model is based on a binary variable for each item from C. The inequalities (2) limit the total consumption for each resource and are called *knapsack constraints*. Example 1 shows a small example instance.

Algorithm 3 Greedy Heuristic for the MDKP

```
1: input: a MDKP instance $\mathcal{I}$
2: $S \leftarrow \emptyset$
3: for $i \leftarrow 1, \ldots, n$ do
4:     if $\left(\sum_{c_j \in S} r_{j,k}\right) + r_{i,k} \leq \mathrm{cap}_k,\ \forall k = 1, \ldots, m$ then
5:         $S \leftarrow S \cup \{c_i\}$
6:     end if
7: end for
8: return $S$
```

3.1.1 Solving Sub-instances to Optimality

Remember that LNS defines a sub-instance based on a partial solution S_{partial} at each iteration. In order to solve such a sub-instance, the following constraints must be added to the ILP model for the MDKP that was outlined above:

$$x_i = 1 \quad \forall\, c_i \in S_{\text{partial}} \tag{4}$$

For solving a sub-instance in the context of CMSA to optimality, the ILP model must be restricted to using items from C', instead of considering the complete set C.

3.1.2 Constructing Solutions to the MDKP

Another algorithmic component needed by LNS and by CMSA is a greedy algorithm. This algorithm is used by LNS in a deterministic way for generating the initial solution. CMSA makes use of this algorithm in a probabilistic way for generating solutions at each iteration. For the description of this greedy heuristic, we henceforth assume that the items in C are ordered with respect to the following *utility values* in a non-increasing way:

$$u_i \leftarrow \frac{p_i}{\sum_{k=1}^{m} r_{i,k}/\mathrm{cap}_k} \qquad \forall\, c_i \in C. \tag{5}$$

To clarify, the items in C are ordered such that $u_1 \geq u_2 \geq \ldots \geq u_n$. The greedy algorithm which is pseudo-coded in Algorithm 3 uses the utility values as a static greedy weight function. More specifically, the greedy heuristic simply adds items—in the order as determined by the utility values—to an initially empty partial solution S until the solution is non-extensible.

While the initial solution of LNS is generated by applying this heuristic as shown in Algorithm 3, CMSA makes use of this heuristic—in function $\mathsf{ProbabilisticSolutionGeneration}(C)$ of line 6 of Algorithm 2—in a probabilistic way. More specifically, CMSA also adds iteratively an item to an initial solution until the solution is non-extensible. At each construction step, an item is chosen as follows. Let S denote the current partial solution and let c_l denote the item that was added to S in the previous construction step. In case $S = \emptyset$, let $l = 0$. In order to choose the next item i^* to be added to S, the first—up to l_{size}—items starting from c_{l+1} that can be added to S with respect to the capacity constraints are added to a candidate list L. Note that the order between the items (as given in C) is maintained in L. Parameter l_{size} is called the *candidate list size*. Then, a real number $\nu \in [0, 1)$ is chosen uniformly at random. In case $\nu \leq d_{\text{rate}}$, i^* is chosen to be the first item from L. Otherwise—that is, in

Example 2: Solution construction step of CMSA for the MDKP

The items from Example 1 are already sorted according to non-increasing utility values. When starting from an empty partial solution $S = \emptyset$, the greedy algorithm (Algorithm 3) adds component c_1 in the first construction step (because it fits into the knapsack). In contrast, assuming that $l_{\text{size}} = 3$, CMSA builds a candidate list L that contains the first $l_{\text{size}} = 3$ items after position $l = 0$ that fit into the knapsack. These are items c_1, c_2, and c_3, that is, $L = [c_1, c_2, c_3]$. Then, with probability d_{rate}, the first item (c_1) from L is chosen and added to S. Otherwise, one of the items from L is chosen uniformly at random. Let us assume that c_2 was added to S. Then, $l := 2$, and the second construction step is started. In this second step, the candidate list is $L = [c_3, c_5, c_9]$. Note that c_4, c_6, c_7, and c_8 would not fit into partial solution $S = \{c_2\}$.

case $\nu > d_{\text{rate}}$—i^* is chosen from L uniformly at random. Just like l_{size}, d_{rate} is an input parameter—called *determinism rate*—of CMSA's solution construction mechanism. In order to clarify this construction procedure, an illustration is provided in Example 2.

3.1.3 Partial Destruction of Solutions in LNS

Finally, it remains to outline the way in which a solution is partially destroyed in LNS. Two variants of a solution destruction mechanism were implemented for this purpose. Given an incumbent solution S, both variants iteratively choose $\max\{3, \lfloor D_r \cdot |S| \rfloor\}$ items from S, and subsequently remove them. In the first variant, the choice of an item to be removed is made at each step uniformly at random. The second variant orders the items in S in an inverse-proportional way according to the utility values, that is, the order of the items in S is inverted with respect to the order of the items in C. Then, at each step, an item from the first $dest_{\text{lsize}}$ items of S is chosen uniformly at random and then removed from S. The parameter that determines which variant is applied is $dest_{\text{type}} \in \{\text{R}, \text{B}\}$, where R stands for random (first variant) and B for biased (second variant).

3.1.4 Example of LNS and CMSA

In order to clarify the different ways in which sub-instances are generated and solved in LNS and CMSA, we illustrate the first iteration of both algorithms in the following for the small MDKP instance from Example 1. This illustration is provided in Example 3.

3.2 Application to the MCSP

The MCSP is a string-based, NP-hard, combinatorial optimisation problem that was defined in the context of genome rearrangement [11]. Given are two input strings s_1 and s_2, both composed of letters from a finite alphabet Σ. Furthermore, the input strings fulfils the property of being *related*, meaning that each letter of Σ has the same number of occurrences in each of the two input strings. Note that the property of being related implies that $|s_1| = |s_2| = n$, that is, the two strings have the same length n. A candidate solution to the MCSP problem is obtained by cutting s_1, respectively s_2, into a set of pieces P_1, respectively P_2. Such a candidate solution (P_1, P_2) is a valid solution, if $P_1 = P_2$. The objective function to be minimised

Example 3: First iteration of LNS and CMSA for the MDKP

LNS: LNS starts by applying the greedy algorithm, which results in the following initial solution: $S_{\text{cur}} = \{c_1, c_2, c_5\}$ (that is, the initial solution includes items 1, 2, and 5)). The objective function value is 49+48+35=132. Now let us assume a destruction rate $D_r = 0.\overline{6}$, using the random destruction mechanism. Accordingly, two out of three items from S_{cur} must be randomly chosen and removed. Let us assume that items c_2 and c_5 are randomly removed, resulting in a partial solution $S_{\text{partial}} = \{c_1\}$. Finally, the ILP solver is applied to solve the resulting sub-instance, using the ILP model for the MDKP as outlined above, with the following additional restriction: $x_1 = 1$, that is, c_1 is forced to form part of any considered solution.

CMSA: CMSA starts with an empty sub-instance $C' = \emptyset$. Let us assume that $n_{\text{a}} = 2$, that is, two solution are generated by the randomised greedy heuristic at each iteration. Moreover, let us assume that the two solutions generated in the first iteration are the following ones:

1. $S_1 = \{c_1, c_5, c_9\}$ with objective function value 49+35+51=135.
2. $S_2 = \{c_2, c_3\}$ with objective function value 48+95=143.

After the construction of these solutions, the items they contain are added to C', resulting in $C' = \{c_1, c_2, c_3, c_5, c_9\}$ and their age values are initialised to zero. Afterwards, the ILP solver is applied to the resulting sub-instance, using the ILP model for the MDKP as outlined above, after replacing all occurrences of C (the complete item set) in this model by C'.

is defined as $f(P_1, P_2) := |P_1|$.[3] Example 4 shows a small example instance.

At this point, it is not obvious why the MCSP belongs to the class of subset selection problems. This will become clear after having introduced the ILP model for this problem (originally proposed in [6]). For this purpose we first introduce the notion of a *common block*. Given s_1 and s_2, a common block $\text{cb}_i = (t_i, k1_i, k2_i)$ consists of a string t_i which is found as a substring of s_1 starting at position $0 \leq k1_i \leq n$ and as a substring of s_2 starting at position $0 \leq k2_i \leq n$. Let C denote the set of all possible common blocks concerning s_1 and s_2. With this definition, the MCSP can be cast as a subset selection problem in the following way. Any subset $S \subset C$ is a candidate solution. Moreover, a candidate solution $S \subset C$ is a *valid solution* if the following conditions are fulfilled:

1. $\sum_{\text{cb}_i \in \mathcal{S}} |t_i| = n$. In other words, summing the length of the substrings corresponding to the common blocks in S must result in n, that is, the length of the input strings.
2. The substrings represented by two common blocks $\text{cb}_i \neq \text{cb}_j \in \mathcal{S}$ neither overlap in s_1 nor in s_2.

With this transformation of the problem, the objective function can be defined as the size of a solution, that is, $f(S) := |S|$ for all valid solutions $S \subset C$. The optimisation goal is

[3]Note that the function could be equivalently defined as $f(P_1, P_2) := |P_2|$.

Example 4: Small MCSP instance

The two input strings are based on alphabet $\Sigma = \{\text{A}, \text{C}, \text{T}, \text{G}\}$.

1. $s_1 = \text{AAGACTG}$
2. $s_2 = \text{ACTAGGA}$

The trivial solution is obtained by cutting both input strings into substrings of length one:

$$P_1 = P_2 = \{\text{A}, \text{A}, \text{A}, \text{C}, \text{T}, \text{G}, \text{G}\}$$

The objective function value is $f(P_1, P_2) = |P_1| = 7$.

The optimal solution of this example instance is obtained by cutting s_1 after the 3^{rd}, the 5^{th} and the 6^{th} letter, and by cutting s_2 after the 1^{st}, the 3^{rd} and the 6^{th} letter:

$$P_1 = P_2 = \{\text{ACT}, \text{AG}, \text{G}, \text{A}\}$$

The objective function value of the optimal solution is $f(P_1, P_2) = |P_1| = 4$.

minimisation. Coming back to the example from Figure **??**, the optimal solution from (c) can be represented as $\{(\text{ACT}, 4, 1), (\text{AG}, 2, 4), (\text{G}, 7, 6), (\text{A}, 1, 7)\}$. With this definition, the ILP model for the MCSP can be expressed as follows.

$$\text{minimise} \quad \sum_{i=1}^{|C|} x_i \tag{6}$$

subject to:

$$\sum_{\text{cb}_i \in C \text{ s.t. } k1_i \le j < k1_i + |t_i|} x_i = 1 \quad \text{for } j = 1, \dots, n \tag{7}$$

$$\sum_{\text{cb}_i \in C \text{ s.t. } k2_i \le j < k2_i + |t_i|} x_i = 1 \quad \text{for } j = 1, \dots, n \tag{8}$$

$$x_i \in \{0, 1\} \qquad \text{for } \text{cb}_i \in C$$

The ILP model consists of exactly one binary variable per common block from C, while the objective function (6) counts the number of selected common blocks. Furthermore, Equations (7) and (8) ensure that (1) each position $j = 1, \dots, n$ of input strings s_1 and s_2 is covered by exactly one selected common block and that (2) selected common blocks do not overlap. Note that, in this ILP model, the number of variables is exponential and the number of constraints is linear in the length of the input strings. A problem instance with $n = 1000$ and $|\Sigma| = 4$, for example, generates an ILP model with $|C| \approx 334.000$ variables.

3.2.1 Solving Sub-instances to Optimality

This is done in exactly the same way as in the case of the MDKP (see Section 3.1.1). The only difference is that, while C is a set of items in the context of the MDKP, it is a set of common blocks in the context of the MCSP problem.

3.2.2 Constructing Solutions to the MCSP Problem

Both LNS and CMSA make use of the following greedy algorithm from [27]. However, before introducing this algorithm we provide the definition of a *valid partial solution.* A subset $S \subset C$ is a valid partial solution, if and only if the substrings represented by two common blocks $\mathrm{cb}_i \neq \mathrm{cb}_j \in \mathcal{S}$ neither overlap in s_1 nor in s_2. Given a valid partial solution $S \subset C$, let $N(S) \subset C$ 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 greedy algorithm starts with an empty partial solution, that is, $S := \emptyset$. At each iteration, one of the common blocks $\mathrm{cb}_i \in N(S)$ is selected such that

$$\mathrm{cb}_i := \mathrm{argmax}\{|t_j| \mid \mathrm{cb}_j \in N(S)\} \ . \tag{9}$$

This block is then added to S, and the process is continued until S is a complete—in the sense of non-extensible—solution. Thus, the algorithm stops once $N(S)$ is empty.

LNS uses this algorithm for generating the initial solution in function GenerateInitialSolution($\mathcal{I}$). Hereby, at each construction step, one of the common blocks that maximise Eqn. 9 is chosen uniformly at random. On the other side, CMSA utilizes this algorithm—in function ProbabilisticSolutionGeneration(C) of line 6 of Algorithm 2—in order to generate solutions in a probabilistic way. Hereby, the choice of a common block $\mathrm{cb}_i \in N(S)$ (where S is the current partial solution under construction) is done as follows. First, the common blocks in $N(S)$ are ordered with respect to the size of the corresponding substrings in a decreasing way. The first $\min\{|N(S)|, l_{\text{size}}\}$ common blocks are stored in the candidate list L. Then, a real number $\nu \in [0, 1)$ is chosen uniformly at random. In case $\nu \leq d_{\text{rate}}$, cb_i is chosen to be the first common block from L. Otherwise—that is, in case $\nu > d_{\text{rate}}$—cb_i is chosen from L uniformly at random. Both l_{size} and d_{rate} are important parameters of CMSA's solution construction mechanism.

3.2.3 Partial Destruction of Solutions in LNS

In the case of the MCSP problem we only implemented the solution destruction mechanism that, given a solution S, randomly removes $\max\{3, \lfloor D_r \cdot |S| \rfloor\}$ common blocks from S. This is because making use of the inverse of the greedy criterion in order to select the common blocks to be removed—as done in the case of the MDKP—does not seem to make sense.

4 Empirical Study

For both problems, the described LNS variants and CMSA were coded in ANSI C++ using GCC 7.4.0 for compilation. The experimental evaluation was performed—in single-threaded mode—on a cluster of computers with "Intel® Xeon® CPU 5670" CPUs of 12 nuclei of 2933 MHz and (in total) 32 Gigabytes of RAM. Moreover, all ILPs in LNS and CMSA were solved with IBM ILOG CPLEX V12.8.

This section is structured as follows. First, the generation of the MDKP and MCSP benchmark instances is described. Second, the parameter tuning process (both concerning CMSA and LNS) is outlined. Finally, the experimental results are provided and analysed in detail.

4.1 Problem instances

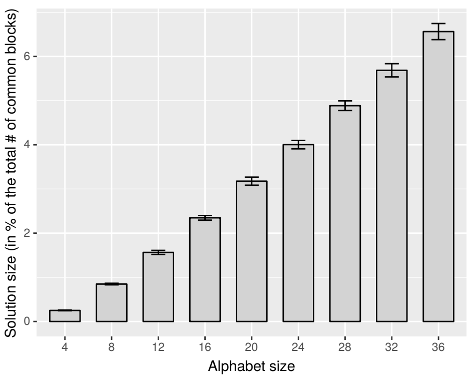
As mentioned before, both the MDKP and the MCSP problem were selected for this study because they are parameterizable. This means that, in both cases, it is possible to produce problem instances from the whole range between instances in which solutions are rather small (that is, instances in which valid solutions contain rather few items, respectively common blocks) and instances in which solutions are rather large (that is, instances in which solutions contain rather many items, respectively common blocks). The type of a problem instance (with respect to the solution size) is determined by the resource capacities in the context of the MDKP, and by the alphabet sizes in the context of the MCSP problem. In particular, low resource capacities (respectively, small alphabet sizes) lead to instances with small solutions, while high resource capacities (respectively, large alphabet sizes) lead to instances with large solutions. This is easy to see in the case of the MDKP. The lower the resource capacities, the less items fit into the knapsack, and vice versa. In the case of the MCSP, instances based on small alphabet sizes lead to much larger sets of common blocks. Therefore, solutions will be rather small—in relation to the size of the set of common blocks—in the context of instances based on rather small alphabets. This claim will be confirmed at the end of this subsection by a set of experiments.

MDKP instances. We used the methodology as described in [25, 12] for the generation of MDKP instances. In particular, five different values for n (the number of items) were considered: $n \in \{100, 500, 1000, 5000, 10000\}$. Moreover, the number of resources (m) was fixed to 30.[4] The *tightness* of a problem instance is determined by the resource capacities. The methodology from [25, 12] allows to determine the instance tightness by means of a parameter α which may take values between zero and one. The lower the value of α—that is, the tighter the generated problem instance—the smaller are the solutions, and vice versa. In order to generate instances over the whole tightness range we chose $\alpha \in \{0.1, 0.2, \ldots, 0.8, 0.9\}$. Finally, the resource requirements $r_{i,j}$ were always chosen uniformly at random from $\{1, \ldots, 1000\}$. In total, 30 instances were generated for each combination of n and α, and the whole benchmark set consists of 1350 problem instances.

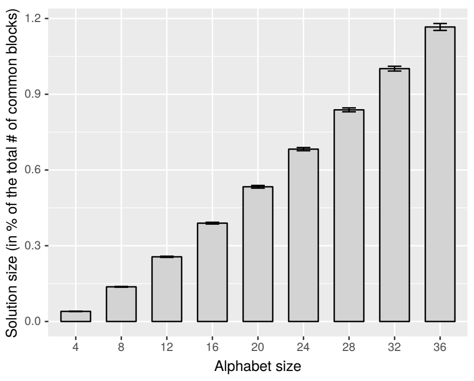
MCSP instances. We considered input string lengths (n) from $\{400, 800, 1200, 1600, 2000\}$ and alphabet sizes ($|\Sigma|$) from $\{4, 8, 12, 16, 20, 24, 28, 32, 36\}$. For each combination of n and $|\Sigma|$, 30 problem instances were randomly generated. This makes again a total of 1350 problem instances. In order to support our claim that instances based on smaller alphabets are characterised by smaller solutions, we applied the greedy algorithm outlined in Section 3.2.2 to all instances with input string lengths $n = \{400, 2000\}$. The graphics in Figure 1 show the resulting solution sizes (in percent with respect to the size of the complete sets of common blocks) in terms of barplots over 30 instances.

Note that all problem instances are provided at `http://iiia.csic.es/~christian.blum/repo_CMSA_vs_LNS/`. The material comes with `README` files that explain the file formats.

[4]Note, in this context, that the complete experimentation was also repeated for instances with $m = 10$ and $m = 50$ resources. The outcome was qualitatively the same as for $m = 30$ resources.



(a) Instances with $n = 400$



(b) Instances with $n = 2000$

Figure 1: Solution sizes (in percent of the total number of common blocks) for the MCSP instances with input string lengths $n = 400$ (a) and $n = 2000$ (b) over the whole range of considered alphabet sizes. The barplots (including the variance) show that solution sizes grow approx. linearly with increasing alphabet size.

4.2 Tuning

The automatic configuration tool irace [35] was used for tuning both algorithms for both problems. In order to be sure to obtain the best possible algorithm performance, irace was applied for each combination of n (number of items, respectively input string length) and α, respectively $|\Sigma|$. More specifically, for each combination of n and α (in the context of the MDKP) and each combination of n and $|\Sigma|$ (in the context of the MCSP problem) we generated three additional random instances for tuning. The budget of irace was set to 1000 algorithm runs. Moreover, the following computation time limits were chosen for both LNS and CMSA:

1. MDKP: 60 CPU seconds for instances with $n = 100$, 120 CPU seconds for those with $n = 500$, 210 CPU seconds for those with $n = 1000$, 360 CPU seconds for those with $n = 5000$, and 600 CPU seconds for those with $n = 10000$.
2. MCSP: 60 CPU seconds for instances with $n = 400$, 120 CPU seconds for those with $n = 800$, 210 CPU seconds for those with $n = 1200$, 360 CPU seconds for those with $n = 1600$, and 600 CPU seconds for those with $n = 2000$.

Parameters of CMSA. In the following we provide a short summary of the parameters of CMSA that are considered for tuning: (1) the number of solution constructions per iteration (n_{a}), (2) the maximum allowed age ($age_{\max}$) of solution components, (3) the determinism rate (d_{rate}), (4) the candidate list size (l_{size}), and (5) the maximum time (in CPU seconds) allowed for CPLEX per application to each sub-instance ($t_{\max}$). The following parameter value ranges were chosen concerning the five parameters of CMSA (for both problems).

- $n_{\mathrm{a}} \in [2, 20]$
- $age_{\max} \in \{1, 5, 10, 20, 50, \inf\}$, where inf means that no item (respectively, common block) is ever removed from the sub-instance.

- $d_{\text{rate}} \in \{0.0, 0.1, \ldots, 0.8, 0.9\}$, where a value of 0.0 means that the selection of the next item (respectively, common block) to be added to the partial solution under construction is always done randomly from the candidate list, while a value of 0.9 means that solution constructions are nearly deterministic.
- $l_{\text{size}} \in [3, 20]$
- $t_{\max} \in \{1, 2, 3, 4, 5, 6\}$ for $n = 100$ (MDKP), resp. $n = 400$ (MCSP), $t_{\max} \in \{1, 2, 4, 6, 8, 10\}$ for $n = 500$ (MDKP), resp. $n = 800$ (MCSP), $t_{\max} \in \{3, 6, 9, 12, 15, 18\}$ for $n = 1000$ (MDKP), resp. $n = 1200$ (MCSP), $t_{\max} \in \{4, 8, 12, 16, 20, 24\}$ for $n = 5000$ (MDKP), resp. $n = 1600$ (MCSP), $t_{\max} \in \{6, 12, 18, 24, 32, 36\}$ for $n = 10000$ (MDKP), resp. $n = 2000$ (MCSP).

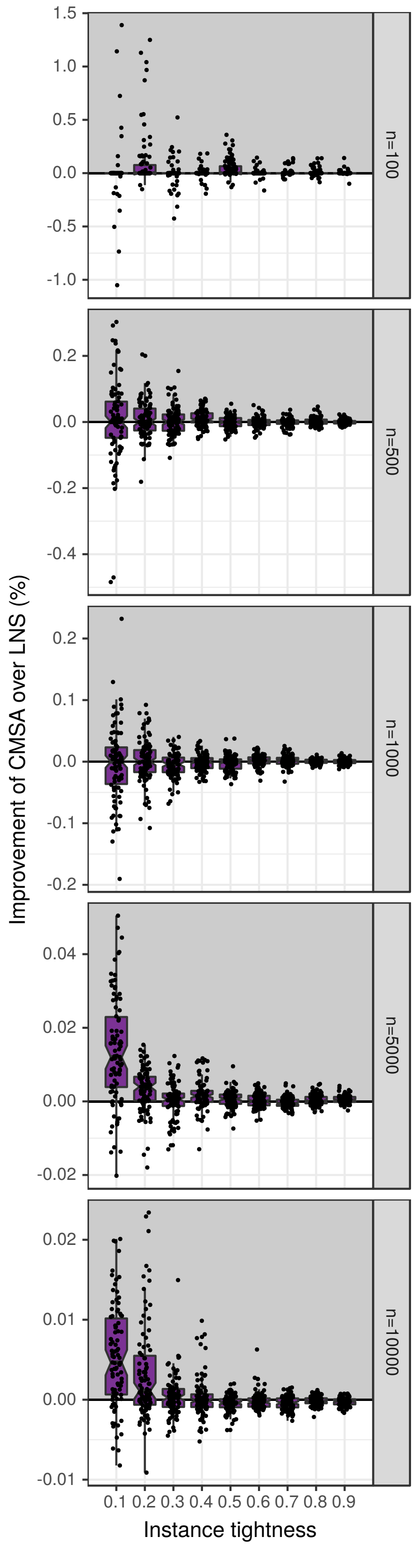
Parameters of LNS. The following LNS parameters were considered for tuning: (1) the lower and upper bounds—that is, D^l and D^u—of the destruction rate, (2) the increment of the destruction rate (D^{inc}) , and (3) the maximum time $t_{\max}$ (in CPU seconds) allowed for CPLEX per application to a sub-instance. Moreover, in the case of the MDKP problem, $dest_{\text{type}}$ (the destruction type) and $dest_{\text{lsize}}$ (the size of the candidate list for destruction) are tuned. The following parameter value ranges were chosen concerning the LNS parameters (for both problems, if not otherwise indicated).

- $D^l, D^u \in \{0.1, 0.2, \ldots, 0.8, 0.9\}$ under the condition that $D^l \leq D^u$. Note that when $D^l = D^u$, the destruction rate D_r is fixed throughout the algorithm run.
- $D^{\text{inc}} \in \{0.01, 0.02, \ldots, 0.08, 0.09\}$
- The value range for $t_{\max}$ was chosen in the same way as for CMSA (see above).
- Only in the case of the MDKP:
  1. $dest_{\text{type}} \in \{\text{R}, \text{B}\}$ (where R stands for random, and B for biased).
  2. $dest_{\text{lsize}} \in \{2, 3, 5, 10\}$.

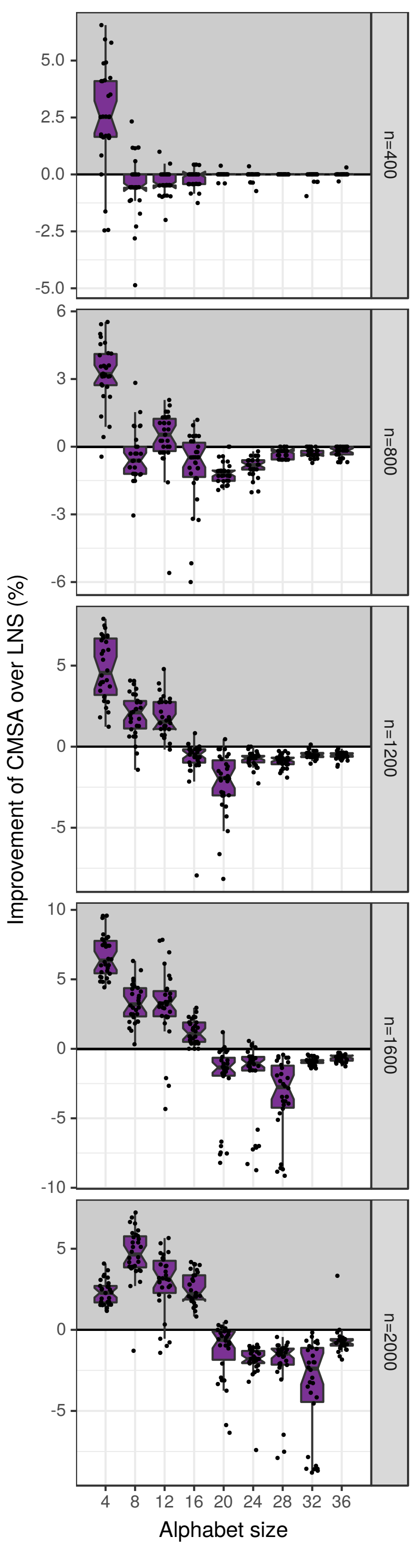
The tuning results—that is, the algorithm parameters that were used for the final experimental evaluation—are provided in Appendix A (MDKP) and Appendix B (MCSP problem).

4.3 Results

Both LNS and CMSA were applied to all 1350 MDKP instances and to all 1350 MCSP instances exactly once, with the computation time limits as indicated in Section 4.2. Figure 2 provides the results for both problems by means of boxplots. In particular, there is exactly one graphic for each value of n, which refers to the number of items in the context of the MDKP, and to the length of the input strings in the context of the MCSP problem. The x-axes of the graphics range from the instances with the smallest solutions on the left, to the instances with the largest solutions on the right. Remember that the solution size is determined by the value of $\alpha \in \{0.1, 0.2, \ldots, 0.8, 0.9\}$ in the context of the MDKP, and by the alphabet size $|\Sigma| \in \{4, 8, \ldots, 32, 36\}$ in the context of the MCSP problem. In the former, α is also called the instance tightness. The boxes in the boxplots of Figure 2 show the improvement of CMSA over LNS (in percent). In other words, when a data point is



(a) Problem: MDKP



(b) Problem: MCSP

Figure 2: Improvement of CMSA over LNS (in percent). Each box shows the differences for the corresponding 30 instances. Note that negative values indicate that LNS obtained a better result than CMSA, and vice versa.

positive (a value greater than zero) CMSA has obtained a better result than LNS, and vice versa. In order to ease the readability of these graphics, the positive area has a shaded background. Note that the output of both LNS and CMSA for each problem instance is provided at `http://iiia.csic.es/~christian.blum/repo_CMSA_vs_LNS/`. The material comes with `README` files that explain the format of the algorithms' output.

The following main observation can be made:

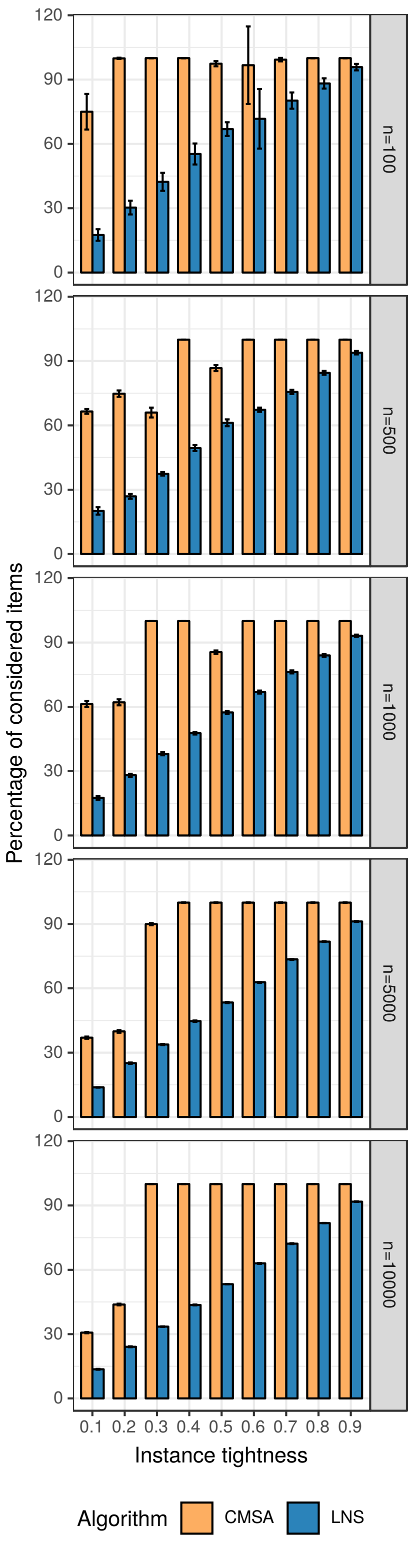
- Concerning the MDKP, the performance of CMSA and LNS is very much comparable for the smallest three instance sizes ($n \in \{100, 500, 1000\}$). However starting from $n = 5000$, a clear advantage of CMSA over LNS arises in the context of instances with small solutions, that is, instances generated with $\alpha \in \{0.1, 0.2\}$. The results concerning the MDKP only partially support our intuition that CMSA generally works better than LNS in the context of problem instances for which solutions are rather small, and that the opposite is the case in the context of problem instances for which solutions are rather large. This is because the second part of the intuition (about the advantage of LNS over CMSA) cannot clearly be identified.
- In contrast to the MDKP, the results concerning the MCSP problem strongly support our intuition. Moreover, as it could have been expected, the empirical support for our intuition becomes stronger and more convincing with growing problem size (input string length).

Finally, note that these observations only hold for the relation of CMSA to the chosen LNS variant. CMSA might behave differently in relation to other LNS variants.

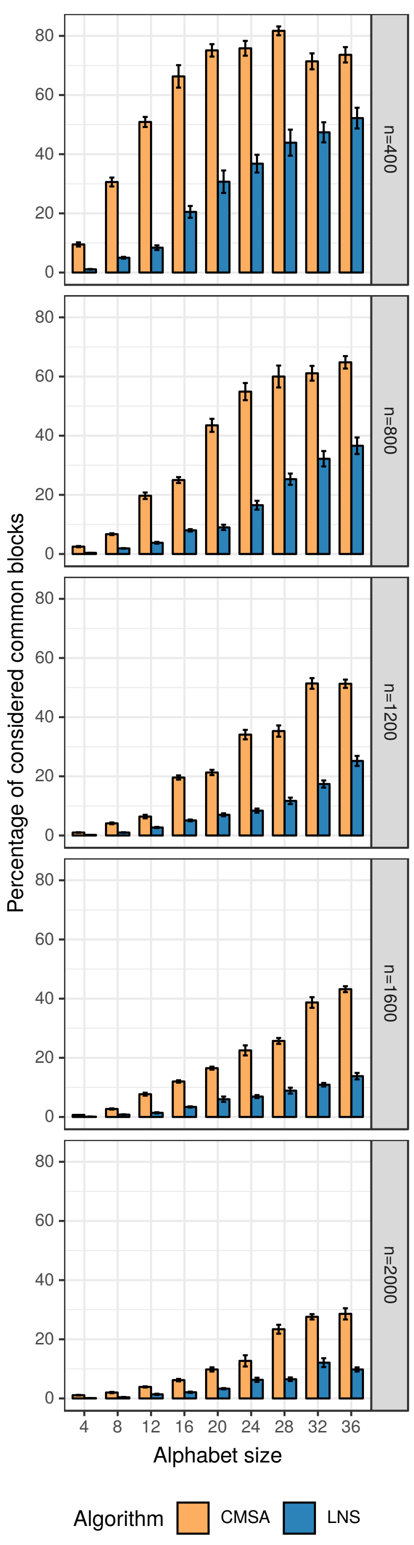
4.4 Analysis of Solution Sizes

In order to shed some further light on the differences between CMSA and LNS, we also measured the percentage of items (in the context of the MDKP), respectively common blocks (in the context of the MCSP), that appeared in at least one of the solutions visited by the algorithm within the allowed computation time. This information is provided in the graphics of Figure 3 by means of barplots. Again, we present one graphic per combination of n and m, respectively n and $|\Sigma|$. In the following we say that the higher this percentage the more explorative is the corresponding algorithm. The following observations can be made:

- First of all, the percentage of used items is—for both problems—always higher for CMSA than for LNS. This means that, with the optimised parameter settings as determined by irace, CMSA is more *explorative* than LNS. This, apparently, pays off in the context of problems with small solutions.
- On the downside, to be explorative does rather not seem to be beneficial—at least not in the context of the MCSP—when problems are characterised by large solutions.
- The difference between the two problems is that, with growing solution size, the percentage of explored items approaches the maximum of 100% in the case of the MDKP, while this is not the case in the context of the MCSP. In order to understand that, remember that the number of common blocks in a MCSP instance is exponential in the instance parameters, while the number of items of an MDKP instance is linear in the problem size.



(a) Problem: MDKP



(b) Problem: MCSP

Figure 3: Percentage of the items (MDKP) and the common blocks (MCSP) that were used in at least one visited solution. Each bar shows the average over the respective 30 problem instances, and the respective standard deviation (see the error bars).

4.5 Analysis of Local Optima Networks

To further understand the differences between the search behaviour of the two algorithms, we studied their search trajectories by means of analysing the corresponding local optima networks. Specifically, we adapted the Monotonic LON (MLON) model [40] to analyse the trajectories of the considered hybrid metaheuristics. Another novelty in this article is the idea of merging the MLON models induced by two different algorithms, CMSA and LNS in our study, in order to compare their trajectories with a graphical support.

4.5.1 Definitions

In order to define a network model, we need to specify the nodes and edges. The relevant definitions are given below, as well as a description of the sampling process to construct the network models.

Nodes. The nodes correspond to (near-)optimal solutions to the considered sub-instances, obtained by the application of the exact solver CPLEX to these sub-instances. Specifically, nodes are the solutions denoted as S'_{opt} after applying the functions Reconstruct and SolveSubinstance in Algorithms 1 and 2 respectively. The set of nodes is denoted as L.

Monotonic Edges. Edges are directed and connect two consecutive solutions in a search trajectory of the studied algorithms. Edges are called *monotonic* as they record only non-deteriorating transitions between nodes (i.e improving or equal evaluation). Specifically, there is a directed edge between solution S_{cur} and solution S'_{opt} (in Algorithms 1 and 2), if S'_{opt} is better than or equal to S_{cur}. Edges are weighted with estimated frequencies of transition. The weight is the number of times a transition between two given nodes occurred during the process of sampling and constructing the MLON. The set of edges is denoted by E.

Monotonic LON (MLON). The MLON is the directed graph MLON $= (L, E)$, with node set L, and monotonic edge set E as defined above.

Sampling and MLON model construction. The MLONs were generated for a representative set of the studied instances. For each of these instances a MLON was constructed by aggregating all the unique nodes and edges encountered across 10 independent runs (search trajectories) of each algorithm. For the MDKP, instances with $n \in \{5000, 10000\}$ (number of items) and $\alpha \in \{0.1, 0.5, 0.9\}$ (tightness) were considered. For the MCSP, instances with $n \in \{1600, 2000\}$ (input string lengths) and $|\Sigma| \in \{4, 20, 36\}$ (alphabet sizes) were used. For each problem and parameter combination, the benchmark set consists of 30 instances.

4.5.2 Network Metrics

For each combination of problem, instance parameters, and algorithm, a MLON model is created. Each MLON aggregates 10 runs of the given algorithm for the given instance. The MLON model is formally defined as a directed graph MLON $= (L, E)$, where L is the set of nodes and E the set of edges. The following metrics were computed for all the generated MLONs.

- *Number of nodes*: The number of nodes in the MLON, which corresponds to the number of unique solutions visited. More formally, this metric measures the cardinality of the set of nodes ($|L|$).
- *Proportion of improving edges*: Computed as the number of edges that are improving (i.e edges that link a solution with inferior quality to one with higher quality) divided by the total number of edges (which, in the MLON model, includes those edges between solutions with equal quality). This measure gives an indication of the amount of neutrality (i.e. solutions of equal evaluation) in the search space. More formally, this metric is computed as the ratio $\frac{|E_i|}{|E|}$, where E_i is the set of edges that are improving, and E is the whole set of edges.
- *Mean Hamming distance*: The average of the pairwise Hamming distances between all consecutive nodes in the MLON. More formally, this metric is computed as

$$\frac{\sum_{j=1}^{|E|} Hamming(s_j, e_j)}{|E|} \tag{10}$$

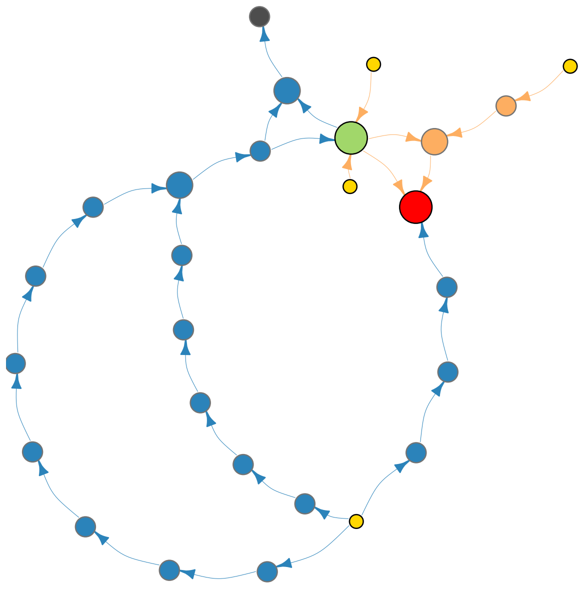
  where s_j, e_j are the incident (start and end) solutions of edge j in the MLON.
- *Hamming distance to best*: Calculates the total pairwise Hamming distance in the shortest path from a starting solution to the solution with the best quality found. More formally, let us assume $s_s, \ldots, s_b$ is the shortest path of length l (measured as the number of edges) between a starting solution s_s in the sampling process, and the best solution found s_b. This metric is computed as: $\sum_{j=1}^{l-1} Hamming(s_j, s_{j+1})$, where s_j and s_{j+1} refer to adjacent solutions in the shortest path.
- *Delta fitness to best*: Calculates the fitness difference between a starting solution and the best solution found. As in general there are several starting solutions, the starting solution that produces the shortest path towards the best solution is considered to compute this metric. More formally, let us assume $s_s, \ldots s_b$ is the shortest path of length l (measured as the number of edges) between a start solution s_s in the sampling process, and the best solution found s_b. This metric is computed as: $|f(s_s) - f(s_b)|$, where f is the objective function.

4.5.3 Network Visualisation

Visualisation plays a fundamental role in network analysis, often revealing structural features that are difficult to asses by computing statistical metrics only. Most previous work on local optima networks visualised the models arising from a single local search neighbourhood (defining the nodes), and a single transition operator (defining the edges). Here, we propose to combine the models arising from the sampling processes of two different algorithms. More formally, let $\text{MLON}_{\text{CMSA}} = (L_{\text{CMSA}}, E_{\text{CMSA}})$ and $\text{MLON}_{\text{LNS}} = (L_{\text{LNS}}, E_{\text{LNS}})$ be the graph models of the two studied algorithms for a given instance. We then construct $\text{MLON}_{\text{merged}}$ as the union of the two graphs. Specifically, $\text{MLON}_{\text{merged}} = (L_{\text{CMSA}} \cup L_{\text{LNS}}, E_{\text{CMSA}} \cup E_{\text{LNS}})$. The merged graph contains nodes and edges which are present in at least one of the algorithm graphs. Attributes are kept for the nodes and edges indicating whether they were visited by both algorithms or by one of them only.

●	Solution visited by CMSA
→	Search transition of CMSA
●	Solution visited by LNS
→	Search transition of LNS
●	Solution visited by both algorithms
●	Solution at the start of a run (either algorithm)
●	Solution at the end of a run (either algorithm)
●	Best solution found across all runs and algorithms
Node Size	Proportional to incoming weighted degree

(a) Visualisation legend



(b) Network visualisation

Figure 4: Merged MLON visualisation for an MDKP instance with $n = 50$, $m = 30$ and tightness $\alpha = 0.5$. The visualisation legend is shown at the top.

To produce network plots we use the R statistics package and the graph layout methods implemented in the igraph library [14]. Specifically, we considered a *force-directed* layout algorithm, Fruchterman-Reignold [21]. Other layout algorithms could be used; for example, when there are no cycles or very few cycles, a tree-layout algorithms can be useful. Example visualisations using the Reingold-Tilford [43] tree layout algorithm for the MCSP problem can be found in the accompanying document on supplementary material. Force-directed layout algorithms are based on physical analogies and do not rely on any assumptions about the structure of the networks. These algorithms strive to satisfy the following generally accepted criteria [21]:

- Vertices are distributed roughly evenly on the plane (a circle in the igraph implementation).
- The number of crossing edges is minimised.
- The lengths of the edges are approximately uniform.
- The inherent symmetries in the networks are respected, i.e., sub-networks with similar inherent structure are usually laid out in a similar manner.

As an example, Figure 4 shows the graph visualisation of the merged MLON model for a small instance of the MDKP problem with $n = 50$, $m = 30$ and tightness $\alpha = 0.5$. For illustrative purposes 3 runs for each algorithm were used to construct the two MLONs in this example. The features of nodes and edges in the merged MLON plot (Fig. 4b) reflect properties of the search dynamics. The size of the nodes is proportional to their incoming weighted degree (also called strength), which indicates how often a node was visited and thus 'attracts' the search process. The nodes and edges visited by only one of the two algorithms are displayed in different colours; light orange for CMSA, and blue for LNS. The initial solutions for all runs are visualised as yellow nodes. The red node illustrates the best solution found. A legend summarising the visualisation features is shown in Figure 4a.

Looking at the network plot (Fig. 4b), we can appreciate that the three LNS runs, visualised in blue, start from the same initial solution (yellow node) at the bottom right. From these three runs, one is successful and reaches the best-found solution (red node), while the other two runs follow different trajectories that converge at the end to the same final solution (dark grey node), indicating that this is an end solution with inferior fitness than the best solution found. The CMSA runs start from three different initial solutions visualised in yellow. All CMSA trajectories converge to the best found solution. The node in green identifies a solution that was visited by both algorithms. The plot illustrates that, in this example, the CMSA trajectories are much shorter (in terms of the number of edges) than the LNS trajectories.

4.5.4 Results of the Network Analysis

The distributions of the network metrics described in section 4.5.2 above, across the 30 instances of each problem and parameter combination, are provided in Figure 5 for the MDKP (left column) and the MCSP (right column). The merged MLONs of representative instances are displayed in graphical form in Figure 6 (for the MDKP) and in Figure 7 (for the MCSP). These network visualisations correspond to the merged MLONs generated by 10 independent runs of both algorithms.

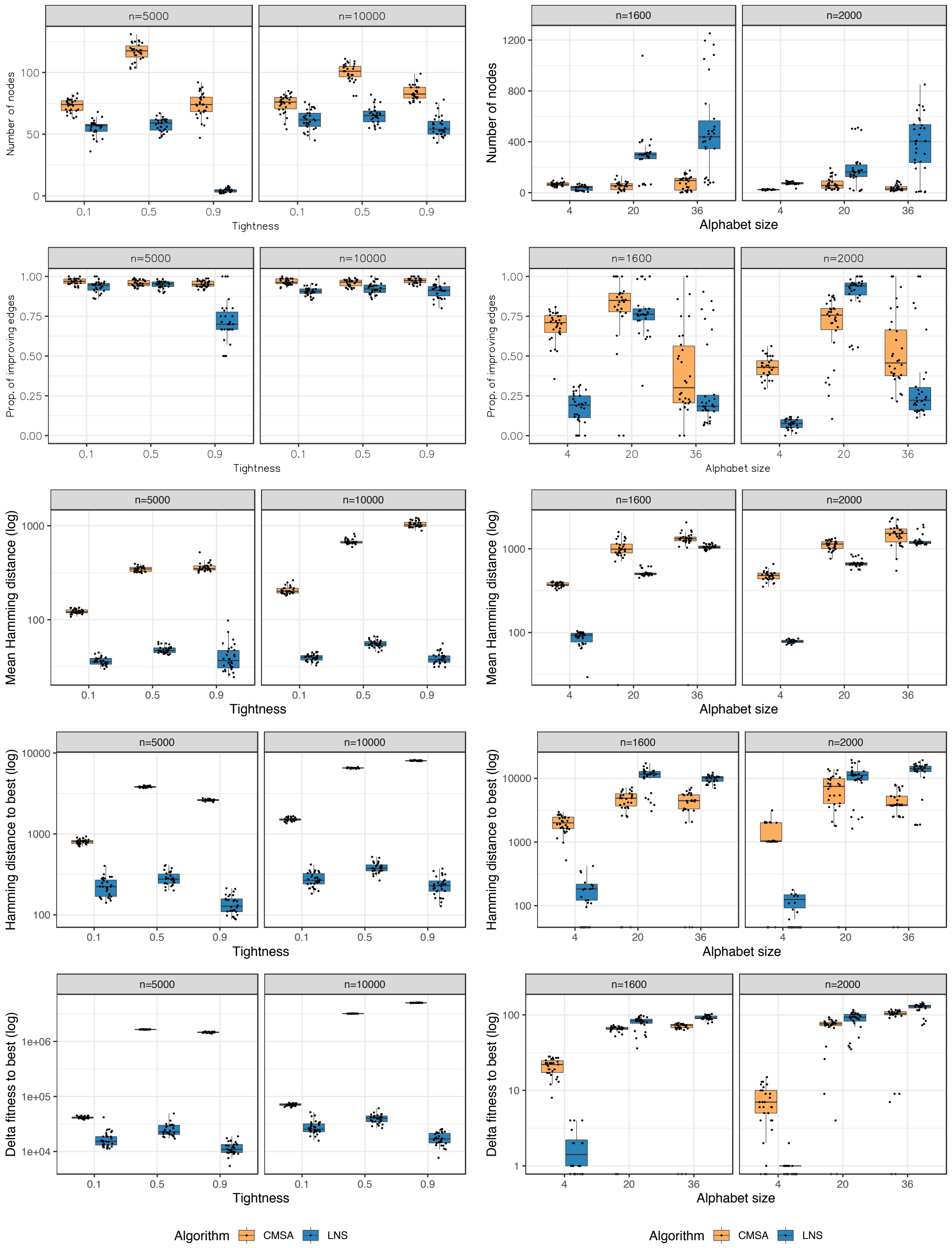


Figure 5: Distribution of MLON metrics. MDKP (left), instances with $n \in \{5000, 10000\}$ (number of items) and $\alpha \in \{0.1, 0.5, 0.9\}$ (tightness). MCSP (right), instances with $n \in \{1600, 2000\}$ (input string lengths) and $|\Sigma| \in \{4, 20, 36\}$ (alphabet sizes).

As discussed above in Section 4.5.3, the features of the nodes and edges in the MLON visualisations (Figures 6 and 7) reflect properties of the search dynamics of the two algorithms. Remember that the visualisation legend is summarised in Fig. 4a. Notice that for the MDKP (Figure 6), the LNS runs start from the same initial deterministic greedy solution, whereas the CMSA runs start from 10 different solutions, that is, the best solutions generated for the creation of the initial sub-instances. Red nodes illustrate the best solution(s) found across all the runs and algorithms. Notice that there might be more than one red node (Fig. 7c), which indicates that several different solutions share the best evaluation found. Solid transition edges indicate improving moves, while dashed edges indicate intended transitions to solutions with equal fitness. The star-like shapes observed in the MCSP MLON for alphabet size 36 (Fig. 7c), indicate solutions with equal fitness that were explored during the search process. Only improving moves are accepted by the algorithms, therefore, the search progresses only when a solution with better fitness is achieved. Finally, nodes in green (Fig. 6) indicate solutions that were visited (shared) by both algorithms in their combined search trajectories, while dark grey nodes represent the end points of trajectories, i.e. those nodes without outgoing edges (having an inferior evaluation than the best solution found).

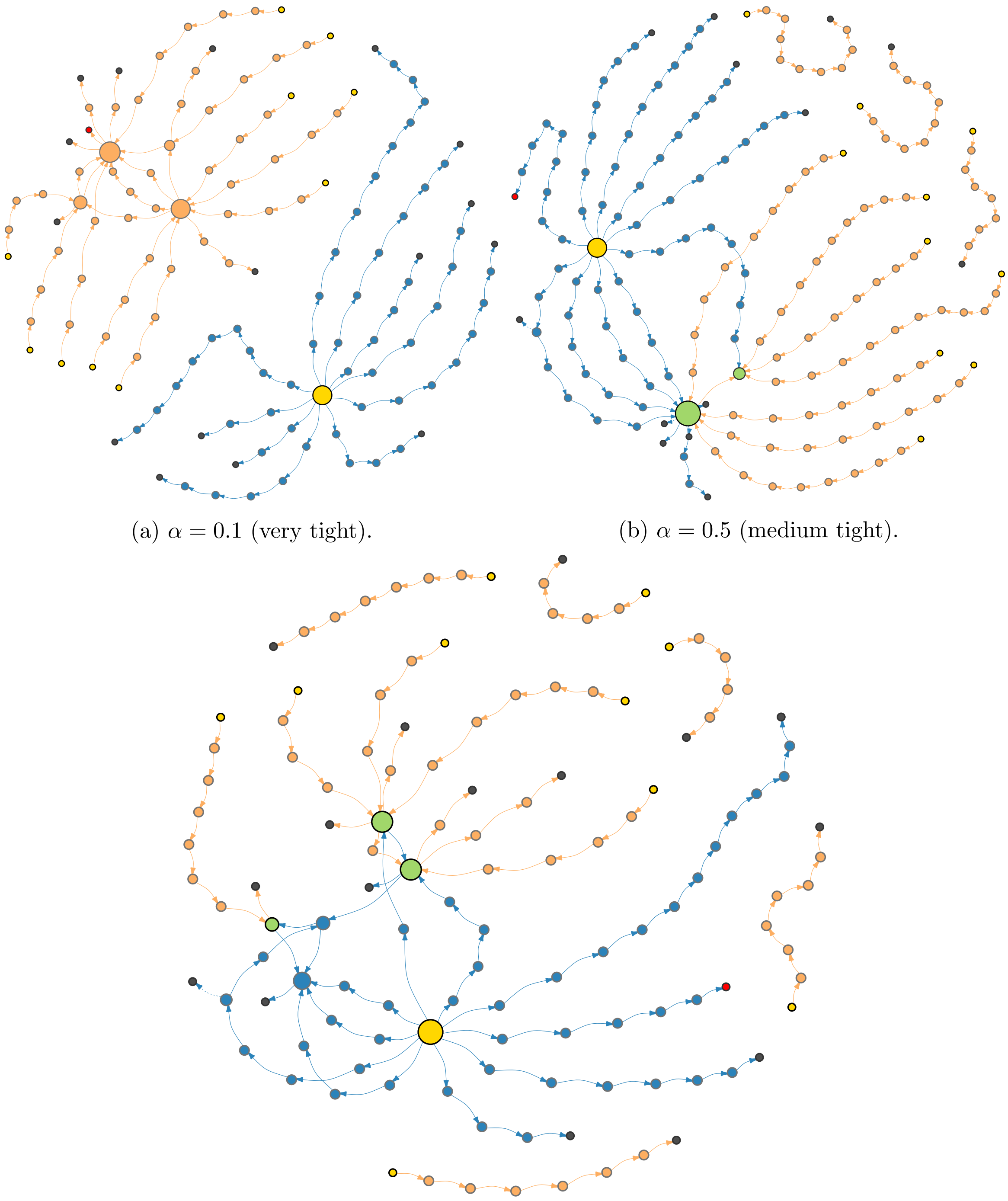
After studying both the MLON metric values for each algorithm and the graphical representations of the merged MLONs, the following observations can be made for the MDKP problem:

- In the case of large and tight MDKP instances—see, for example, Figure 6a for the case $n = 10000$ and $\alpha = 0.1$—LNS seems, in a sense, to be lost, because all 10 runs (1) lead to different final solutions, (2) there is no overlap of the trajectories of the 10 runs, and (3) there is no overlap with the 10 trajectories of CMSA.[5] On the other side, the 10 CMSA trajectories seem to be attracted by high-quality solutions (as indicated by the overlap of the 10 trajectories in later stages of the search). As shown in the bottom graphic of Figure 2a, CMSA generally outperforms LNS in this case. This analysis is supported by the fact that (1) the mean Hamming distance between successive solutions in the trajectories of CMSA is higher than in the case of LNS, and (2) the number of nodes in the MLONs of CMSA is higher than the one in the MLONs of LNS (see Figure 5).

- The analysis described above for MDKP instances with $n = 10000$ and $\alpha = 0.1$ becomes inverted when moving to instances with $\alpha = 0.5$ (see Figure 6b) and $\alpha = 0.9$ (see Figure 6c). The search trajectories of CMSA and LNS for $\alpha = 0.5$ (medium tight instances) look quite similar, showing some overlap between them (green nodes in Figures 6 (b) and (c)). For instances with $\alpha = 0.9$ (non-tight instances), the overlap between the 10 LNS trajectories is clearly higher than the overlap between the CMSA trajectories (seen as nodes of increased size). Moreover, the LNS trajectories are, on average, clearly longer.

In summary, even though our initial intuition was only partially supported by the results obtained for the MDKP, the differences that are shown in the MLON metrics and graphics clearly indicate the different search behaviour of both algorithms for what concerns tight vs. non-tight instances. Moreover, one reason why the MDKP results do not fully support our intuition might be grounded in the fact that it is rather easy to find high-quality solutions

[5]In this context, note that all 10 LNS runs start from the same initial solution, because a deterministic greedy heuristic is used to generate the initial LNS solution.

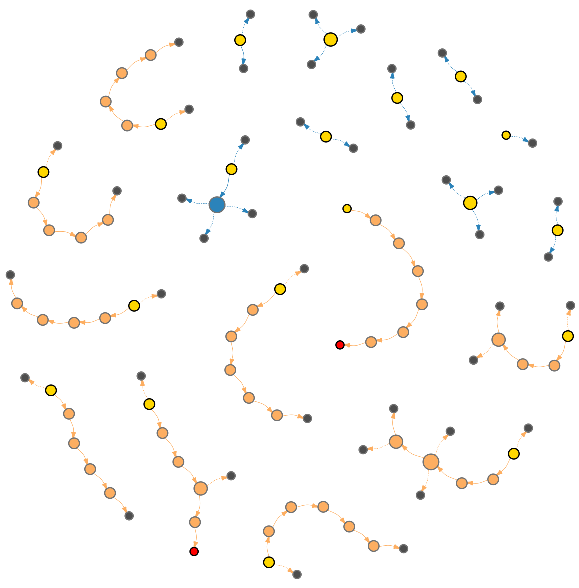


(a) $\alpha = 0.1$ (very tight).

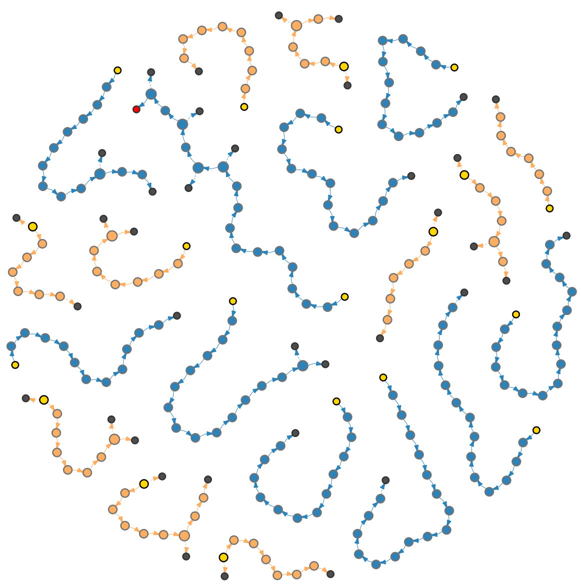
(b) $\alpha = 0.5$ (medium tight).

(c) $\alpha = 0.9$ (non-tight).

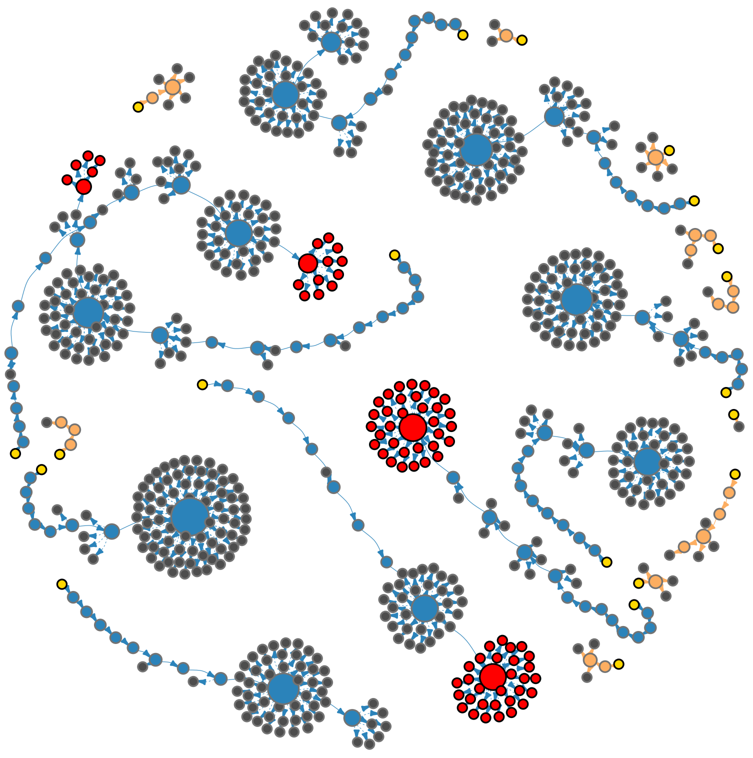
Figure 6: Merged MLONs for MDKP instances with $n = 10000$ and three values of instance tightness α.



(a) $|\Sigma| = 4$ (small solutions).



(b) $|\Sigma| = 20$ (medium size solutions).



(c) $|\Sigma| = 36$ (large solutions).

Figure 7: Merged MLON for an MCSP instance with $n = 1600$ and three values of alphabet size $|\Sigma|$.

for MDKP instances, while it is not easy at all to find the best (or near-optimal) solutions.

Next, studying the merged MLONs as well as the metric values of the individual MLONs for the MCSP problem allows to draw the following conclusions:

- First, remember that the results obtained for the MCSP problem fully supported our intuition about the relative performance of LNS and CMSA over the range between instances with small solutions and instances with large solutions. The clear picture shown by the results in Figure 2b is largely reflected by the metric values displayed in Figure 5 (see the left column of graphics). The *Hamming distance to best*, for example, is smaller for the MLONs corresponding to LNS for instances with $|\Sigma| = 4$ (small solutions), while this is generally inverted for $|\Sigma| \in \{20, 36\}$. The same holds for *Delta fitness to best*.

- Concerning the merged MLONs in Figure 7, the change of algorithm behaviour can clearly be observed when moving from the case with small solutions (Figure 7a), over the case with medium-size solutions (Figure 7b), to the case with large solutions (Figure 7b). In the first case, the search trajectories of CMSA are substantially longer (in terms of the number of steps) than those of LNS. Moreover, this also holds for the total Hamming distance from the initial solutions to the final solutions of the search trajectories (see measure *Hamming distance to best* in Figure 5). In fact, LNS has often difficulties in finding a solution that improves over the initial Greedy solution. One of the reasons for this is the fact that LNS does not really solve a reduced ILP model. It solves the original problem in which some of the variables have fixed values. However, for instances with small solutions this concerns only very few variables. In contrast, the ILP model solved by CMSA at each iteration is—especially in the case of instances with small solutions—a substantially reduced version of the original ILP model.

- The situation described for instances with $|\Sigma| = 4$ (small solutions) is already reversed when moving to the case with $|\Sigma| = 20|$ (medium-size solutions). Here, LNS generally outperforms CMSA. The search trajectories of LNS are longer both in terms of the number of steps, and in terms of the total Hamming distance from the initial solution to the end of a trajectory. Finally, in the case of large solutions (case $|\Sigma| = 36$), LNS performs clearly better than CMSA, as indicated by much longer search trajectories. In fact, CMSA has difficulties to improve over the solutions found in the first iteration. The reason for this might be as follows. In general, the larger the sub-instances in CMSA (that is, the more components they contain) the better is the chance that the sub-instances contain improving solutions. Therefore, the tuning procedure adjusts the parameters for each instance type such that the size of the sub-instances is (1) large enough to contain improving solutions and is (2) still small enough in order to be solved efficiently by the ILP solver. With growing solution size (which implies also a growing sub-instance size) this becomes more and more difficult. This is indicated by the increase of the value of parameter d_{rate} (determinism rate) with growing alphabet size (see Table 4b). This means that, with growing alphabet size, the CMSA algorithm is forced to work with an increasing value of d_{rate} which causes the sub-instance size to decrease and, at the same time, the chances to find an improving solution in the sub-instance to decrease as well. Finally, note that the search space in the case $|\Sigma| = 36$ is characterised by a high degree of neutrality, which is indicated by the clusters of solutions (remember that our merged MLON graphics show both, transitions to better solutions, and non-accepted transitions to solutions with the same quality than the current solution).

Finally, in addition to observing the changing behaviour of both CMSA and LNS over the range of different instances (from those with small solutions to those with large solutions) both in the MLON metrics and in the merged MLON graphics, it is also possible to observe differences between the algorithms that are maintained over all considered instances. This concerns, for example, the *Mean Hamming distance* of the transitions in the studied MLONs. CMSA shows always a higher mean Hamming distance than LNS, that is, CMSA is generally able to perform larger jumps in the search space. This seems to pay off in the case of instances with rather small solutions, while this does not seem to be the case in for instances with rather large solutions.

In summary, we can state that our initial intuition about the differences in algorithm behaviour between CMSA and LNS are supported by the obtained experimental results. Moreover, merged MLON graphics and the metrics of the individual MLONs are useful tools for the (comparative) study of algorithm behaviour.

4.6 Algorithm Performance without Specialised Tuning

The results presented so far were based on a fine-grained tuning of both algorithms for each combination of instance size and instance tightness (in the case of the MDKP), respectively alphabet size (in the case of the MCSP problem). This was done in order to obtain the best possible performance of each algorithm for each case. However, in practise algorithms are generally tuned for larger subsets of instances. In order to see if the relative performance of the two algorithms changes when adopting this general practise, we decided (1) to produce one single parameter setting per algorithm and problem (MDKP and MCSP problem), and (2) to repeat the experimentation from Figure 2. For the purpose of tuning we first used again irace with the same parameter value domains as outlined in Section 4.2. A total of 2000 algorithm runs were used as budget in irace for each of the four tuning runs (combination of algorithm and problem). This global tuning resulted in the following parameter value settings for the MDKP:

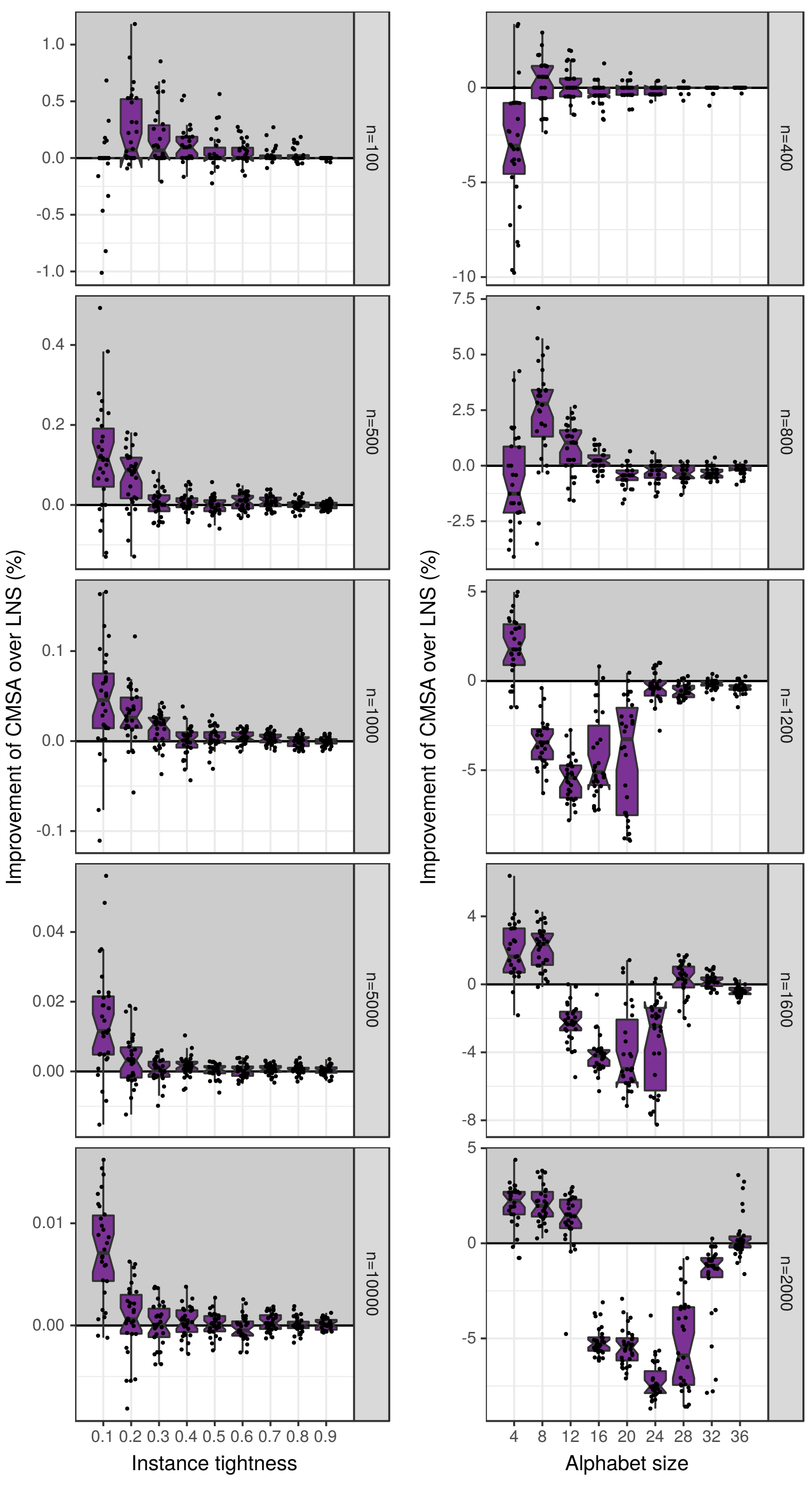
- LNS: $D^l = 0.7$, $D^u = 0.8$, $D^{\text{inc}} = 0.02$, $dest_{\text{type}} = \text{R}$, $t_{\max} = 12.0$.
- CMSA: $n_{\text{a}} = 9$, $age_{\max} = 1$, $d_{\text{rate}} = 0.6$, $l_{\text{size}} = 11$, $t_{\max} = 4.0$.

Moreover, the following parameter values were determined by irace for the application to the MCSP problem:

- LNS: $D^l = 0.5$, $D^u = 0.8$, $D^{\text{inc}} = 0.09$, $t_{\max} = 20.0$.
- CMSA: $n_{\text{a}} = 16$, $age_{\max} = 50$, $d_{\text{rate}} = 0.2$, $l_{\text{size}} = 15$, $t_{\max} = 24.0$.

Both algorithms were then applied with these parameter values to all problem instances, and the relative performance of both algorithms is shown in Figure 8, in the same way as the results obtained with fine-grained tuning from Figure 2.

Curiously, in the case of the MDKP, the observations from Figure 2a (fine-grained tuning) also hold—in an even stronger way—for Figure 8a (global tuning). For what concerns



(a) Problem: MDKP (b) Problem: MCSP

Figure 8: Improvement of CMSA over LNS (in percent) after global tuning (that is, using the same parameter setting for all instances). Each box shows the differences for the corresponding 30 instances. Note that negative values indicate that LNS obtained a better result than CMSA, and vice versa.

the results for the MCSP problem, the observations from Figure 2b still hold (with a few exceptions) for most alphabet sizes. However, the results for instances with alphabet size 4 (instances with the smallest solutions) have changed, especially for what concerns the smaller problem instances ($n \in \{400, 800\}$). The reason for this might be that, generally, the parameter value settings required by both algorithms for the MCSP problem are highly dependent on the alphabet size. Therefore, when allowing only one single parameter value setting, it is only natural that the algorithms' performance decreases/changes for certain ranges of the alphabet size.

4.7 Comparison to the Current State of the Art

Even though the main aim of this paper is not the comparison to the current state of the art concerning the MDKP and MCSP problems, we also want to shed some light on this aspect. In the case of the MCSP problem, note that the considered CMSA algorithm is currently the state-of-the-art approach [7].[6] Therefore, no additional experiments are required in the case of the MCSP problem.

In order to demonstrate the performance of both CMSA and LNS for the MDKP, both algorithms were applied with a time limit of 200 seconds per run to all 30 benchmark instances with 500 items and 10 resources from the OR-Library (`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`). These instances, which are from the set by Chu and Beasley, are generally said to belong to the most difficult ones used in the literature. Each algorithm was applied 100 times to each problem instance in order to be comparable to the results from [30], which is one of the most recent works from the literature. Moreover, the parameter settings obtained by global tuning (see Section 4.6) were used for this purpose. The results of CMSA and LNS are compared with the ones of the DQPSO algorithm from [30] and the TPTEA algorithm from [31]. Both DQPSO and TPTEA can currently be regarded as state-of-the-art metaheuristic MDKP solvers. The results are presented in Table 1. The first column contains the instance name. The next four columns (with the general heading "best") provide the value of the best solution found by each of the four algorithms within 100 runs. Furthermore, the following four columns (with the general heading "average") contain the average solution quality obtained over 100 runs. And finally, the last four columns (with the general heading "average time") show the average computation times needed by each algorithm to find the best solutions of each run. As a general conclusion, it can be stated that both CMSA and LNS obtain results that are comparable to the state of the art. In fact, both algorithms have a slightly better *best-performance* than current state-of-the-art algorithms, while their *average performance* is slightly worse. In terms of computation time, both LNS and CMSA are comparable to DQPSO, and much faster than TPTEA.

5 Conclusions and Future Work

In this work we have performed a comparative analysis of two hybrid algorithms for combinatorial optimisation: (1) a variant from the well-established family of large neighbourhood search algorithms, and (2) construct, merge, solve & adapt, which is a more recent method. Both algorithms work in an iterative way, solving reduced problem instances at each iteration. The main difference between the two algorithms is to be found in the way in which

[6]The aim of the follow-up work from [4] was to make CMSA applicable to large-scale problem instances.

Table 1: Comparison of CMSA and LNS to the current state-of-the-art metaheuristic solvers for the MDKP.

Instance	best				average				average time			
	TPTEA	DQPSO	CMSA	LNS	TPTEA	DQPSO	CMSA	LNS	TPTEA	DQPSO	CMSA	LNS
10.500.0	117801	117779	117799	**117809**	117736.17	**117754.82**	117752.4	117745.40	3664.7	105.3	88.5	121.5
10.500.1	119200	**119206**	119198	119189	119137.47	**119179.74**	119150.9	119153.51	3354.9	152.7	97.0	106.45
10.500.2	119159	**119215**	119211	119211	119108.27	119162.61	**119170.2**	119153.40	4469.7	127.8	96.2	117.5
10.500.3	**118829**	118813	118825	118813	**118793.93**	118777.36	118784.7	118786.09	3150.1	39.8	65.4	116.98
10.500.4	116456	**116509**	**116509**	**116509**	116405.17	**116460.83**	116450.6	116449.66	3582.2	146.2	81.4	102.99
10.500.5	**119483**	119470	119466	119472	119441.80	119435.57	**119453.9**	119446.08	3646.9	157.4	71.9	104.85
10.500.6	119775	**119827**	119810	119790	119739.70	**119782.76**	119770.1	119763.83	3789.5	134.1	75.6	108.43
10.500.7	**118323**	118320	118309	**118323**	118258.27	**118265.30**	118258.0	118258.27	4228.5	146.7	89.0	114.77
10.500.8	117801	117781	**117815**	117781	117705.97	**117763.24**	117732.8	117739.69	3168.7	122.3	101.0	112.16
10.500.9	119196	**119212**	119210	**119212**	119161.90	119166.54	**119174.4**	119164.65	3338.8	124.7	92.8	111.74
10.500.10	217351	217365	**217377**	**217377**	217313.67	**217326.03**	217320.4	217318.80	3984.4	76.7	39.3	104.82
10.500.11	219059	219063	**219077**	**219077**	219022.70	219027.54	**219034.6**	219033.37	3598.2	136.0	74.4	110.55
10.500.12	**217847**	217847	217792	217793	**217786.73**	217760.63	217763.7	217768.94	4010.4	173.1	40.5	102
10.500.13	**216868**	216843	**216868**	**216868**	**216836.33**	216824.05	216830.2	216807.69	3403.7	125.3	96.8	104.17
10.500.14	213814	213843	**213859**	**213859**	213780.27	213817.08	**213825.4**	213814.62	4095.0	134.4	90.1	108.03
10.500.15	**215086**	215062	215075	215075	**215049.57**	215034.68	215032.9	215025.06	3622.3	166.4	91.8	110.34
10.500.16	217926	**217931**	**217931**	**217931**	217884.80	217884.36	**217891.1**	217880.38	4281.3	181.3	94.2	104.64
10.500.17	**219984**	**219984**	**219984**	**219984**	219947.37	**219965.07**	219951.7	219945.48	3908.2	102.0	68.8	107.06
10.500.18	214363	**214382**	214375	214375	214327.43	**214337.14**	214316.3	214314.62	3999.5	165.9	91.8	122.46
10.500.19	**220887**	220865	220886	220882	**220864.43**	220847.10	220847.6	220836.26	3211.7	152.5	84.9	109.46
10.500.20	**304387**	**304387**	304353	304363	**304364.47**	304354.01	304344.2	304341.36	3025.1	67.4	26.5	99.43
10.500.21	**302379**	302358	**302379**	302370	**302364.47**	302346.04	302335.3	302332.24	2883.6	98.6	81.6	103.64
10.500.22	**302416**	302408	**302416**	**302416**	302398.13	302399.10	**302401.4**	302382.15	3497.9	77.4	90.7	110.6
10.500.23	**300784**	**300784**	300747	300757	**300758.80**	300745.46	300742.5	300733.41	967.0	58.6	48.9	113.76
10.500.24	**304374**	**304374**	304366	304366	**304361.13**	304353.75	304344.1	304346.12	3509.5	112.6	57.9	88.21
10.500.25	**301796**	301766	301767	**301796**	301740.43	**301752.48**	301745.5	301745.67	3808.8	106.4	92.3	105.74
10.500.26	**304952**	304949	**304952**	304949	**304952.00**	304949.00	304949.0	304944.99	3132.6	16.2	14.2	86.17
10.500.27	**296478**	296459	296459	296459	**296455.53**	296444.16	296451.0	296444.47	3138.7	153.6	79.5	113.48
10.500.28	**301359**	301357	301357	301357	**301349.27**	301332.24	301318.7	301319.80	3187.4	163.1	101.8	106.95
10.500.29	**307089**	**307089**	**307089**	**307089**	**307088.23**	307068.83	307059.9	307054.91	2351.9	83.1	90.9	112.92
average	212840.70	212841.60	**212842.03**	212841.73	212804.48	**212810.58**	212806.78	212801.70	3467.04	120.26	**76.71**	108.06

these reduced problem instances are generated.

Our computational study in the context of two subset selection problems has supported our intuition about the relative behaviour of the two algorithms. This intuition states that construct, merge, solve & adapt would generally outperform the chosen large neighbourhood search variant in the context of problem instances with rather small solutions, and that the opposite would be the case when instances with large solutions are concerned. While this intuition was only partially supported by the results for the multidimensional knapsack problem, it was fully supported by the results obtained for the minimum common string partition problem. Note that this holds with respect to the chosen large neighbourhood search variant. The relative behaviour of construct, merge, solve & adapt and other large neighborhood search variants might be different.

The comparative study of the two algorithms was supported by the study of merged local optima networks that graphically show the behaviour of 10 runs of both algorithms for single problem instances, and by metric values computed from the individual local optimal networks. This way of comparing algorithm behaviour has been found to be very useful for the interpretation of the obtained results. In fact, a further study of merged local optimal networks is, in our opinion, the most promising avenue for future work. We are convinced that this type of graphical tools will play an important role in future studies on algorithm behaviour.

Acknowledgements

This work was supported by project CI-SUSTAIN funded by the Spanish Ministry of Science and Innovation (PID2019-104156GB-I00).

References

[1] R. K. Ahuja, J. B. Orlin, and D. Sharma. Very large-scale neighborhood search. *International Transactions in Operational Research*, 7(4-5):301–317, 2000.

[2] E. Angelelli, R. Mansini, and M. G. Speranza. Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, 37(11):2017–2026, 2010.

[3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP. Technical report, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.

[4] C. Blum. Minimum common string partition: on solving large-scale problem instances. *International Transactions in Operational Research*, 27(1):91–111, 2020.

[5] C. Blum and M. J. Blesa. Hybrid techniques based on solving reduced problem instances for a longest common subsequence problem. *Applied Soft Computing*, 62:15–28, 2018.

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

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

[8] C. Blum and G. R. Raidl. *Hybrid Metaheuristics – Powerful Tools for Optimization.* Springer, 2016.

[9] M. A. Boschetti, V. Maniezzo, M. Roffilli, and A. Bolufé Röhler. Matheuristics: Optimization, simulation and control. In M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, and A. Schaerf, editors, *Proceedings of HM 2009 – 6th International Workshop on Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*, pages 171–177. Springer Berlin Heidelberg, 2009.

[10] M. Caserta and S. Voß. A corridor method based hybrid algorithm for redundancy allocation. *Journal of Heuristics*, 22(4):405–429, 2016.

[11] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Computing the assignment of orthologous genes via genome rearrangement. In Y.-P. P. Chen and L. Wong, editors, *Proceedings of APBC 2005 – 3rd Asia Pacific Bioinformatics Conference*, Series on Advances in Bioinformatics and Computational Biology, pages 363–378, 2005.

[12] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Discrete Applied Mathematics*, 49(1):189–212, 1994.

[13] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[14] G. Csárdi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[15] F. Daolio, S. Vérel, G. Ochoa, and M. Tomassini. Local optima networks of the quadratic assignment problem. In *Proceedings of CEC 2010 – IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE Press, 2010.

[16] E. Demir, T. Bektaş, and G. Laporte. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2):346–359, 2012.

[17] R. Dias Saraiva, N. Nepomuceno, and P. Rogério Pinheiro. A two-phase approach for single container loading with weakly heterogeneous boxes. *Algorithms*, 12:67, 2019.

[18] J. P. K. Doye. The network topology of a potential energy landscape: a static scale-free network. *Physical Review Letters*, 88(23), 2002. Article number 238701.

[19] M. Eskandarpour, P. Dejax, and O. Péton. A large neighborhood search heuristic for supply chain network design. *Computers & Operations Research*, 80:23–37, 2017.

[20] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 2003.

[21] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[22] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996.

[23] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. In R. Fleischer and G. Trippen, editors, *Proceedings of ISAAC 2004 – International Symposium on Algorithms and Computation*, number 3341 in Lecture Notes in Computer Science, pages 484–495. Springer Berlin Heidelberg, 2005.

[24] G. Guastaroba and M. G. Speranza. Kernel search: An application to the index tracking problem. *European Journal of Operational Research*, 217(1):54–68, 2012.

[25] S. Hanafi and A. Freville. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106(2-3):659–675, 1998.

[26] P. Hansen and N. Mladenović. Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[27] D. He. A novel greedy algorithm for the minimum common string partition problem. In I. Mandoiu and A. Zelikovsky, editors, *Proceedings of ISBRA 2007 – Third International Symposium on Bioinformatics Research and Applications*, volume 4463 of *Lecture Notes in Computer Science*, pages 441–452. Springer Berlin Heidelberg, 2007.

[28] H. Kellerer, U. Pferschy, and D. Pisinger. *Multidimensional Knapsack Problems*, pages 235–283. Springer Berlin Heidelberg, 2004.

[29] X. Kong, L. Gao, H. Ouyang, and S. Li. Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. *Computers & Operations Research*, 63:7–22, 2015.

[30] X. Lai, J.-K. Hao, Z.-H. Fu, and D. Yue. Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. *Expert Systems with Applications*, 149, 2020. Article number 113310.

[31] X. Lai, J.-K. Hao, F. Glover, and Z. Lü. A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem. *Information Sciences*, 436:282–301, 2018.

[32] E. Lalla-Ruiz and S. Voß. POPMUSIC as a matheuristic for the berth allocation problem. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):173–189, 2016.

[33] S. C. H. Leung, D. Zhang, C. Zhou, and T. Wu. A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack problem. *Computers and Operations Research*, 39(1):64–73, 2012.

[34] E. Lizárraga, M. J. Blesa, and C. Blum. Construct, merge, solve and adapt versus large neighborhood search for solving the multi-dimensional knapsack problem: Which one works better when? In B. Hu and M. López-Ibáñez, editors, *Proceedings of EvoCOP 2017 – 17th European Conference on Evolutionary Computation in Combinatorial Optimization*, number 10197 in Lecture Notes in Computer Science, pages 60–74. Springer International Publishing, 2017.

[35] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[36] N. V. Nepomuceno, P. R. Pinheiro, and A. L. V. Coelho. Tackling the container loading problem: A hybrid approach based on integer linear programming and genetic algorithms. In C. Cotta and J. van Hemert, editors, *Proceedings of EvoCOP 2007 – 7th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 154–165. Springer, 2007.

[37] G. Ochoa and S. Herrmann. Perturbation strength and the global structure of QAP fitnes landscapes. In A. Auger, C. M. Fonseca, N. Lourenco, P. Machado, L. Paquete, and D. Whitley, editors, *Proceedings of PPSN XV – Parallel Problem Solving from Nature*, volume 11102 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 2018.

[38] G. Ochoa, M. Tomassini, S. Vérel, and C. Darabos. A study of NK landscapes' basins and local optima networks. In *Proceedings of GECCO 2008 – Genetic and Evolutionary Computation Conference*, pages 555–562. ACM Press, NY, 2008.

[39] G. Ochoa and N. Veerapen. Mapping the global structure of TSP fitness landscapes. *Journal of Heuristics*, 24(3):265–294, 2018.

[40] G. Ochoa, N. Veerapen, F. Daolio, and M. Tomassini. Understanding phase transitions with local optima networks: Number partitioning as a case study. In B. Hu and M. López-Ibáñez, editors, *Proceedings of EvoCOP 2017 – Evolutionary Computation in Combinatorial Optimization*, volume 10197 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2017.

[41] D. Pisinger and S. Røpke. Large Neighborhood Search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 399–419. Springer US, 2010.

[42] K. Pruhs and G. J. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382(2):151–156, 2007.

[43] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, Mar. 1981.

[44] D. R. Santos-Peñate, C. M. Campos-Rodríguez, and J. A. Moreno-Pérez. A kernel search matheuristic to solve the discrete leader-follower location problem. *Networks and Spatial Economics*, 51:1–26, 2019.

[45] V. Schmid. Hybrid large neighborhood search for the bus rapid transit route design problem. *European Journal of Operational Research*, 238(2):427–437, 2014.

[46] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

[47] E. Talbi, editor. *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*. Springer, 2013.

[48] S. Verel, G. Ochoa, and M. Tomassini. Local optima networks of NK landscapes with neutrality. *IEEE Transactions on Evolutionary Computation*, 15(6):783–797, 2011.

Appendix A: Tuning results for the MDKP

Table 2: Parameter settings of CMSA concerning the MDKP.

(a) MDKP instances with $n \in \{100, 500, 1000\}$ (n refers to the number of items).

α	$n = 100$					$n = 500$					$n = 1000$				
	n_a	age_{max}	d_{rate}	l_{size}	t_{max}	n_a	age_{max}	d_{rate}	l_{size}	t_{max}	n_a	age_{max}	d_{rate}	l_{size}	t_{max}
0.1	8	20	0.5	10	5.0	17	5	0.5	12	8.0	17	5	0.4	8	3.0
0.2	8	1	0.2	9	2.0	11	1	0.8	16	8.0	5	1	0.8	14	9.0
0.3	5	5	0.5	19	3.0	14	10	0.4	3	6.0	10	1	0.5	17	6.0
0.4	4	1	0.7	10	2.0	13	1	0.4	19	1.0	10	1	0.6	18	6.0
0.5	7	1	0.7	4	1.0	16	1	0.9	8	4.0	19	10	0.5	3	12.0
0.6	12	1	0.2	15	2.0	8	1	0.5	17	2.0	3	5	0.4	10	3.0
0.7	8	10	0.8	3	1.0	11	1	0.3	20	2.0	4	5	0.4	20	3.0
0.8	11	5	0.1	17	4.0	4	1	0.6	10	1.0	7	1	0.5	10	3.0
0.9	8	5	0.0	13	2.0	7	5	0.2	20	1.0	3	5	0.4	9	3.0

(b) MDKP instances with $n \in \{5000, 10000\}$ (n refers to the number of items).

α	$n = 5000$					$n = 10000$				
	n_a	age_{max}	d_{rate}	l_{size}	t_{max}	n_a	age_{max}	d_{rate}	l_{size}	t_{max}
0.1	6	1	0.3	6	4.0	4	5	0.7	11	12.0
0.2	3	1	0.3	3	4.0	5	1	0.7	7	6.0
0.3	4	5	0.6	10	8.0	6	1	0.2	11	6.0
0.4	10	1	0.0	9	4.0	8	1	0.6	16	6.0
0.5	9	1	0.1	15	4.0	11	1	0.2	15	6.0
0.6	13	1	0.0	20	4.0	8	1	0.7	19	18.0
0.7	4	5	0.6	17	8.0	6	5	0.5	16	6.0
0.8	4	5	0.6	12	8.0	10	5	0.1	15	12.0
0.9	4	1	0.8	10	4.0	9	1	0.5	13	12.0

Table 3: Parameter settings of LNS concerning the MDKP.

(a) MDKP instances with $n \in \{100, 500, 1000\}$ (n refers to the number of items).

α	$n = 100$					$n = 500$					$n = 1000$				
	D^l	D^u	D^{inc}	$dest_{type}$	t_{max}	D^l	D^u	D^{inc}	$dest_{type}$	t_{max}	D^l	D^u	D^{inc}	$dest_{type}$	t_{max}
0.1	0.9	0.9	–	R	5.0	0.7	0.8	0.06	R	2.0	0.7	0.9	0.06	R	3.0
0.2	0.7	0.9	0.02	R	6.0	0.8	0.8	–	B (5)	10.0	0.5	0.9	0.02	R	3.0
0.3	0.9	0.9	–	R	3.0	0.8	0.8	–	R	8.0	0.8	0.8	–	R	3.0
0.4	0.8	0.9	0.03	R	1.0	0.6	0.8	0.03	R	2.0	0.6	0.9	0.01	R	3.0
0.5	0.2	0.7	0.08	R	1.0	0.7	0.7	–	R	1.0	0.8	0.8	–	R	3.0
0.6	0.7	0.8	0.08	R	1.0	0.7	0.7	–	R	4.0	0.8	0.9	0.01	R	3.0
0.7	0.2	0.5	0.09	B (5)	3.0	0.1	0.6	0.06	B (5)	6.0	0.7	0.8	0.01	R	3.0
0.8	0.6	0.8	0.01	R	4.0	0.2	0.2	–	B (5)	10.0	0.1	0.9	0.05	B (10)	6.0
0.9	0.2	0.8	0.06	R	4.0	0.1	0.4	0.04	B (2)	8.0	0.3	0.5	0.01	B (3)	9.0

(b) MDKP instances with $n \in \{5000, 10000\}$ (n refers to the number of items).

α	$n = 5000$					$n = 10000$				
	D^l	D^u	D^{inc}	$dest_{type}$	t_{max}	D^l	D^u	D^{inc}	$dest_{type}$	t_{max}
0.1	0.7	0.9	0.04	R	8.0	0.7	0.9	0.03	R	6.0
0.2	0.5	0.9	0.03	R	4.0	0.8	0.8	–	R	6.0
0.3	0.8	0.9	0.03	R	12.0	0.7	0.9	0.08	R	12.0
0.4	0.6	0.9	0.06	R	4.0	0.6	0.9	0.08	R	6.0
0.5	0.8	0.8	–	R	8.0	0.7	0.9	0.04	R	6.0
0.6	0.8	0.9	0.07	R	12.0	0.7	0.7	–	R	6.0
0.7	0.7	0.9	0.05	R	4.0	0.7	0.7	–	R	12.0
0.8	0.1	0.8	0.03	B (3)	16.0	0.7	0.9	0.04	R	12.0
0.9	0.3	0.9	0.03	B (5)	24.0	0.8	0.9	0.03	R	6.0

Appendix B: Tuning results for the MCSP problem.

Table 4: Parameter settings of CMSA concerning the MCSP problem.

(a) MCSP instances with $n \in \{400, 800, 1200\}$ (n refers to the input string length).

$\|\Sigma\|$	$n = 400$					$n = 800$					$n = 1200$				
	n_a	$age_{\max}$	d_{rate}	l_{size}	$t_{\max}$	n_a	$age_{\max}$	d_{rate}	l_{size}	$t_{\max}$	n_a	$age_{\max}$	d_{rate}	l_{size}	$t_{\max}$
4	18	10	0.6	5	6.0	18	5	0.1	2	10.0	17	10	0.0	2	12.0
8	17	10	0.5	19	4.0	7	20	0.0	5	10.0	10	10	0.0	5	15.0
12	19	10	0.0	20	4.0	7	5	0.1	10	4.0	4	5	0.1	5	18.0
16	18	20	0.0	20	3.0	16	50	0.8	17	10.0	18	1	0.0	13	15.0
20	14	20	0.0	18	3.0	20	20	0.6	20	8.0	9	5	0.2	15	15.0
24	9	50	0.1	20	5.0	18	10	0.2	19	10.0	11	5	0.4	18	9.0
28	16	50	0.0	19	1.0	16	inf	0.0	20	2.0	19	5	0.8	15	12.0
32	8	1	0.3	14	3.0	17	50	0.1	19	6.0	15	20	0.3	20	15.0
36	7	5	0.1	14	6.0	20	inf	0.2	19	4.0	16	20	0.4	19	6.0

(b) MCSP instances with $n \in \{1600, 2000\}$ (n refers to the input string length).

$\|\Sigma\|$	$n = 1600$					$n = 2000$				
	n_a	$age_{\max}$	d_{rate}	l_{size}	$t_{\max}$	n_a	$age_{\max}$	d_{rate}	l_{size}	$t_{\max}$
4	5	10	0.0	3	20.0	10	inf	0.5	4	12.0
8	11	inf	0.0	4	24.0	9	5	0.0	4	36.0
12	16	1	0.0	9	24.0	4	10	0.0	9	36.0
16	14	1	0.0	9	20.0	5	5	0.0	7	36.0
20	13	1	0.1	11	24.0	4	5	0.0	12	32.0
24	17	1	0.1	11	24.0	3	5	0.0	8	32.0
28	17	10	0.8	19	24.0	11	1	0.1	14	36.0
32	18	5	0.4	20	20.0	12	5	0.3	15	32.0
36	19	10	0.7	18	8.0	14	inf	0.8	19	36.0

Table 5: Parameter settings of LNS concerning the MCSP problem.

(a) MCSP instances with $n \in \{400, 800, 1200\}$ (n refers to the input string length).

$\|\Sigma\|$	$n = 400$				$n = 800$				$n = 1200$			
	D^l	D^u	D^{inc}	$t_{\max}$	D^l	D^u	D^{inc}	$t_{\max}$	D^l	D^u	D^{inc}	$t_{\max}$
4	0.6	0.7	0.09	6.0	0.5	0.5	–	10.0	0.4	0.4	–	18.0
8	0.7	0.9	0.07	6.0	0.6	0.9	0.01	10.0	0.5	0.6	0.06	15.0
12	0.9	0.9	–	4.0	0.7	0.7	–	8.0	0.5	0.8	0.05	18.0
16	0.7	0.9	0.05	2.0	0.7	0.9	0.04	10.0	0.6	0.7	0.05	12.0
20	0.7	0.9	0.01	1.0	0.9	0.9	–	10.0	0.7	0.9	0.01	18.0
24	0.7	0.9	0.07	6.0	0.9	0.9	–	6.0	0.8	0.9	0.05	18.0
28	0.8	0.8	–	3.0	0.8	0.9	0.02	6.0	0.9	0.9	–	15.0
32	0.8	0.8	–	4.0	0.7	0.9	0.03	10.0	0.9	0.9	–	18.0
36	0.5	0.6	0.07	3.0	0.9	0.9	–	6.0	0.9	0.9	–	6.0

(b) MCSP instances with $n \in \{1600, 2000\}$ (n refers to the input string length).

$\|\Sigma\|$	$n = 1600$				$n = 2000$			
	D^l	D^u	D^{inc}	$t_{\max}$	D^l	D^u	D^{inc}	$t_{\max}$
4	0.3	0.4	0.05	16.0	0.2	0.3	0.08	32.0
8	0.4	0.7	0.02	24.0	0.4	0.4	–	36.0
12	0.5	0.8	0.03	20.0	0.4	0.4	–	18.0
16	0.5	0.5	–	12.0	0.5	0.6	0.08	32.0
20	0.6	0.6	–	24.0	0.6	0.6	–	36.0
24	0.7	0.9	0.06	20.0	0.6	0.7	0.05	18.0
28	0.8	0.8	–	24.0	0.7	0.7	–	36.0
32	0.9	0.9	–	20.0	0.7	0.8	0.09	36.0
36	0.9	0.9	–	20.0	0.9	0.9	–	36.0