Self-Adaptive CMSA for Solving the Multidimensional Multi-Way Number Partitioning Problem

Marko Djukanović[a,1], Aleksandar Kartelj[b], Christian Blum[c]

[a] *Faculty of Natural Sciences and Mathematics, University of Banja Luka, Bosnia and Herzegovina*
[b] *Faculty of Mathematics, University of Belgrade, Serbia*
[c] *Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC), Spain*

Abstract

The multidimensional multi-way number partitioning problem takes as input a set of n vectors with a fixed dimension $m \geq 2$, and aims to find a partitioning of this set into $k \geq 2$ non-empty subsets, such that the sums per coordinate across all subsets are as similar as possible. The problem has applications in key encryption, multiprocessor scheduling, minimization of circuit size and delay, and clustering. This paper employs a hybrid meta-heuristic, a self-adaptive version of the CMSA algorithm called Adapt-Cmsa, to solve the problem. Adapt-Cmsa is additionally equipped with an efficient local search procedure that accelerates the convergence towards promising regions of the search space. A comprehensive experimental evaluation shows that Adapt-Cmsa improves over all four competing algorithms from the related literature, especially when it comes to instances with higher k-values, i.e. $k \geq 3$. The observed average relative differences are for several instance groups larger than 25% in favor of Adapt-Cmsa compared to the second-best approach. In fact, a statistical evaluation indicates that Adapt-Cmsa performs significantly better than the other approaches on all instances with $k \geq 3$.

Keywords: Number partitioning problem, hybrid metaheuristics, MILP models, local search
2010 MSC: 68W25, 68W20

URL: `marko.djukanovic@pmf.unibl.org` (Marko Djukanović), `aleksandar.kartelj@gmail.com` (Aleksandar Kartelj), `christian.blum@csic.es` (Christian Blum)

[1]Corresponding author

1. Introduction

Given a set of n m-dimensional vectors, the *multidimensional multiway number partitioning problem* (MDMWNPP) is concerned with finding a partitioning of the set of n vectors into k disjoint subsets—that is, a k-partitioning ($k \geq 2$)—such that the sums of the values at each coordinate for all subsets are balanced. Concrete examples of practical applications of the MDMWNPP are the following ones:

- Partitioning an area containing cities into regions such that an equilibrium among the region's characteristics—including population, economic power, etc—is obtained [26].
- Forming commissions (teams) of experts (people) such that, in terms of certain competencies, the formed commissions are similar [39].
- Public-key cryptography [1].

Other real-world applications of this problem are described in [26, 16]. The number partitioning problem (NPP) is a well-studied variant of the more general MDMWNPP where $m = 1$, $k = 2$ and vectors contain integers (or decimal numbers). This problem is $\mathcal{NP}$–hard in case the size of input numbers is exponential in n [21]. In fact, the NPP is one of Karp's twenty one basic $\mathcal{NP}$–hard problems [8]. Moreover, its decision variant is $\mathcal{NP}$-complete. Practical applications of the NPP include multiprocessor scheduling [29], public key cryptography [20], and the assignment of tasks in low-power application-specific integrated circuits [38]. The NPP has been intensively studied within the last few decades. The efficient *differencing method* for solving the NPP was proposed by Karmarkar and Karp [36]. This work attracted a wide range of researchers to solve the NPP. Note that there exists a tight relation between the NPP and quadratic programming, since the former can be expressed as an unconstrained quadratic binary program and solved efficiently by appropriate black–box solvers [25]. Moreover, many efficient meta-heuristics were proposed, especially aimed at tackling large-sized instances. Examples of such meta-heuristic approaches are a tabu search from [25], and a simulated annealing approach from [30], just to name a few. Two incomplete tree search methods to solve the NPP were introduced in [28]. An exact tree-search method was proposed in [10]. Some randomized approaches to solve the NPP were considered in [33]. A deep computational analysis on solving the NPP can be found in [15].

The MDMWNPP with $m = 1$ and arbitrary large values for n and k was studied in [9, 27, 11]. The first generalization w.r.t. parameter m, thus considering vectors instead of numbers as input, was proposed by Kojic in [12] under the assumption of a fixed value $k = 2$. More precisely, a mixed integer linear programming (MILP) formulation was introduced in that paper. Further, Kratica et al. [23] proposed two meta-heuristics to solve the MDTWNPP. Rodrigez et al. [37] proposed GRASP with exterior path relinking and restricted local search. A genetic algorithm and simulated annealing were designed by Hacirbeyoglu et al. [39]. Santucci et al. [32] proposed a memetic algebraic differential evolution algorithm for binary search spaces (MADEB) to solve the MDTWNPP. They were able to show that their algorithm outperformed other state-of-the art competitors. Santucci et al. [31] further improved their MADEB algorithm, replacing it with a more robust version—called iMADEB—utilizing a self-adaptive algebraic differential mutation scheme on the basis of the Lévy flight concept [13]. This allowed an automatic regulation of the trade-off between exploration and exploitation during the search, thus yielding a new state-of-the-art algorithm for the MDTWNPP.

The most general version of the MDMWNPP considered in this paper, with arbitrary value for m and k, was introduced by Pop and Matei in [2]. The authors proposed a genetic/memetic algorithm (GA) to tackle the problem. Another meta-heuristic method—that is, a variable neighborhood search (VNS)—was introduced by Faria et al. [17]. Concerning exact approaches, recently, two MILP formulations were proposed: one was introduced by Faria et al. [16] and the other one by Nikolic et al. [26]. The former MILP model performs better than the latter one when k is small, while the latter one is better for larger values of k. We emphasize that, in [17], a comparison between VNS and the GA approach from [2] is only partially provided, because the two approaches are compared only on a subset of the standard benchmark set used in the literature. In particular, type A instances are used, while the remaining instance types (B, C, and D) were not considered. Therefore, conclusions for the remaining instance types cannot be drawn from the existing literature. Moreover, the two algorithms were implemented in different programming languages and they were executed on significantly different computer hardware.

1.1. Contributions of this work

The main contributions of this work can be summarized as follows:

1. Since the MDMWNPP is fairly well studied in the context of mathematical programming—that is, there are two existing MILP models—we propose a MILP-based hybrid meta-heuristic called self-adaptive "Construct, Merge, Solve and Adapt" (CMSA) algorithm to solve the MDMWNPP. This algorithm is henceforth labelled Adapt-Cmsa. Even

though Adapt-Cmsa is usually an algorithm that exclusively relies on the step-by-step construction of solutions, we found it beneficial to improve generated solutions by means of an efficient local search procedure, which enforces a stronger search intensification.

2. We provide a comprehensive and fair comparison among all known exact and meta-heuristic approaches from literature concerning the MDMWNPP, including our own Adapt-Cmsa approach. Before executing all algorithms on the same computer and with the same resources, we actually re-implemented all algorithms in the same programming language (C++). All source codes are publicly available.

3. A thorough statistical evaluation is performed on the basis of the obtained results, and appropriate conclusions are drawn. In short, the Adapt-Cmsa approach is able to significantly outperform all competing algorithms in terms of the obtained solution quality when applied to problem instances with moderate and large values of k, while GA performs strongly in the context of instances with $k = 2$.

This work is organized as follows. In Section 2 we present the model of the search space of the MDMWNPP. Section 3 presents the chosen MILP model from the literature whose role is significant for the quality of our algorithm. Moreover, we describe our re-implementations of the two meta-heuristic approaches from the literature. In Section 4 the proposed Adapt-Cmsa framework and its application to solve the MDMWNPP are described in detail. Section 5 reports on the exhaustive numerical and statistical comparison on the basis of the obtained results. Conclusions and an outline of future work are provided in Section 6.

2. Problem definition and search space

Let $S = \{v_1, v_2, \ldots, v_n\}$ be a set of n vectors of a fixed length $m \in \mathbb{N}$, that is, $v_i = (v_{i,1}, \ldots, v_{i,m}) \in \mathbb{R}^m$, for all $i = 1, \ldots, n$. Moreover, let $k \in \{2, \ldots, n\}$ be a fixed number. The MDMWNPP asks for finding a k-partitioning $\mathcal{S} = (S_1, \ldots, S_k)$ of set S that minimizes the following objective function:

$$f(\mathcal{S}) = \max_{p_1,p_2 \in \{1,\ldots,k\} | p_1 > p_2, j \in \{1,\ldots,m\}} \left| \sum_{v_i \in S_{p_1}} v_{i,j} - \sum_{v_i \in S_{p_2}} v_{i,j} \right| \tag{1}$$

In this context, note that $\mathcal{S}$ being a k-partitioning implies that $\bigcup_{i \in \{1,\ldots,k\}} S_i = S$ and $S_{p_1} \cap S_{p_2} = \emptyset$, for each $p_1, p_2 \in \{1, ..., k\}, p_1 \neq p_2$. Any given MDMWNPP instance is henceforth denoted by (S, n, m, k).

Note that this problem can be modelled as follows. The search space can be defined as the set of all n-dimensional integer vectors $s = (s_1, \ldots, s_n)$, $1 \leq s_i \leq k$ for each $i \in \{1, \ldots, n\}$. Hereby, $s_i = j$ means that vector v_i is assigned to partition S_j. Moreover, s is a feasible solution iff for each $j \in \{1, \ldots, k\}$ there is at least one $i \in \{1, \ldots, n\}$ such that $s_i = j$, i.e. there are no empty partitions. With

$$\max_{x,y \in S} |x - y| = \max_{x \in S} x - \min_{x \in S} x,$$

the objective function (1) can then be rewritten in a more convenient way:

$$f(\mathcal{S}) = \max_{l \in \{1,\ldots,m\}} \left(\max_{j \in \{1,\ldots,k\}} \left(\sum_{v_i \in S_j} v_{il} \right) - \min_{j \in \{1,\ldots,k\}} \left(\sum_{v_i \in S_j} v_{il} \right) \right) \tag{2}$$

Note that there is an obvious symmetry in this definition of the search space of the MDMWNPP. For example, solutions $(1, 1, 2)$ and $(2, 2, 1)$ represent the same feasible solution for an example instance with $n = 3$ and $k = 2$. In order to eliminate redundant considerations of symmetric solutions, we apply the following *symmetry breaking* rule. We only consider solutions $\mathcal{S} = (S_1, \ldots, S_k)$ such that, for all $i < j$, the vector with the lowest index in S_i has a lower index than the vector with the lowest index in S_j. In the above example solution $(1, 1, 2)$, the lowest index of the vectors that belong to S_1 is 1, while the lowest index of vectors in S_2 is 3. In other words, this solution is considered. On the other hand, solution $(2, 2, 1)$ is not considered since the lowest index of the vectors belonging to S_1 is three, while the lowest index of the vectors belonging to S_2 is one. Note that solution $(2, 2, 1)$ can easily be rearranged and transformed into solution $(1, 1, 2)$. This also holds for any arbitrary solution, and the computational cost of the rearrangement is $O(n)$. Consequently, vector v_1 always belongs to S_1.

3. Approaches from the literature

In this section we will describe three MDMWNPP approaches from the literature which are relevant for our Adapt-Cmsa approach as well as for comparison purposes. In particular, we will describe the utilized MILP model and two meta-heuristic approaches, namely Vns and Ga, in the way in which they were re-implemented. This re-implementation was necessary since source codes or binaries are not publicly available for these meta-heuristic approaches from the literature. The basic principle that we followed in our re-implementations is as follows: in case of several ways to re-implement a certain aspect of a method (when the textual description from the original paper was not clear) we do it in a way that does not degrade the quality or efficiency of the method. Details on our re-implementations are exhaustively provided in the next sections, while source codes are made publicly available at `https://github.com/kartelj/adapt-cmsa-mdmwnpp`.

3.1. MILP model

In this section, we present the MILP model from [26]—labelled Coam—that is used internally in Adapt-Cmsa. The second available MILP model from [16] is not described here, even though later it is included in the experimental evaluation.

The following variables are being used by Coam:

- Binary variables $x_{ij},\ i \in \{1, ..., n\},\ j \in \{1, ..., k\}$, such that $x_{ij} = 1$ if vector v_i is included in partition S_j, otherwise $x_{ij} = 0$.
- Real variables $y_l, z_l,\ l \in \{1, ..., m\}$ denote the maximum, respectively minimum, sums of the numbers in the l-th coordinate across all partitions.
- A positive real variable r that corresponds to the objective function:

$$r = \max_{l \in \{1,\ldots,m\}} \{y_l - z_l\}.$$

The Coam model can then be stated as follows:

$$\min \quad r \tag{3}$$

$$s.t.$$

$$\sum_{j=1}^{k} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, n\} \tag{4}$$

$$\sum_{i=1}^{n} x_{ij} \geq 1, \qquad \forall j \in \{1, \ldots, k\} \tag{5}$$

$$\sum_{i=1}^{n} v_{il} \cdot x_{ij} \leq y_l, \qquad \forall j \in \{1, \ldots, k\}, \forall l \in \{1, \ldots, m\} \tag{6}$$

$$\sum_{i=1}^{n} v_{il} \cdot x_{ij} \geq z_l, \qquad \forall j \in \{1, \ldots, k\}, \forall l \in \{1, \ldots, m\} \tag{7}$$

$$y_l - z_l \leq r, \qquad \forall l \in \{1,\dots,m\} \tag{8}$$
$$x_{ij} \in \{0,1\}, \qquad \forall i \in \{1,\dots,n\}, \forall j \in \{1,\dots,k\} \tag{9}$$
$$y_l, z_l \in \mathbb{R}, \qquad \forall l \in \{1,\dots,m\} \tag{10}$$
$$r \in [0,+\infty) \tag{11}$$

Constraints (4) ensure that each vector belongs to exactly one partition. Constraints (5) ensure that neither partition is empty. Constraints (6)–(8) calculate the objective function whose value is stored in variable r.

3.2. Re-implementation of the VNS algorithm for the MDMWNPP

The VNS algorithm for the MDMWNPP, proposed in [17], had to be re-implemented since neither the source code nor binaries were available on the internet. In addition, we were not able to obtain them from the authors. The re-implementation was done in accordance with the VNS algorithm description, which consequently produced results very similar to those obtained by the authors of the VNS approach.

Algorithm 1 provides the pseudo-code of Vns for the MDMWNPP. The algorithm consists of the following steps:

- First, the initial (and currently best) solution s_{bsf} is constructed by using a simple greedy method, whose details can be found in [17].
- After that, the algorithm iterates until the pre-defined CPU time limit is reached. Each iteration consists of the following steps:
  - Shaking procedure in the neighborhood of size r, where parameter r_{max} sets the maximal value of r. The authors of VNS in [17] used $r_{max} = 3$ in their work. This means that r vectors are randomly selected from partitions of s_{bsf}. Each one of these vectors is subsequently moved to a randomly chosen partition, thus producing new candidate solution s_{new}.
  - Afterwards, a local search procedure named `LS1best()` is applied to s_{new}.
  - Depending on the value of r, additional local searches `LS2best()` or `LS3best()` are applied.
  - If the new solution s_{new}, after application of shaking and the local search procedure(s), is better than s_{bsf}, then s_{new} becomes the current best solution and the shaking neighborhood size is reset to 1.

Algorithm 1 VNS for MDMWNPP

```
1: Input: a problem instance $\mathcal{I} = (S, n, m, k)$, $r_{max}$ – max. neighborhood size
2: Output: $s_{bsf}$, the best solution found within the time limit
3: $s_{bsf} \leftarrow$InitializeSolution($\mathcal{I}$)
4: $r \leftarrow 1$
5: while time limit is not reached do
6:     $s_{new} \leftarrow$ Shake($s_{bsf}$, $r$)
7:     $s_{new} \leftarrow$ LS1best($s_{new}$)
8:     if $r = 2$ then
9:         $s_{new} \leftarrow$ LS2best($s_{new}$)
10:    else if $r = 3$ then
11:        $s_{new} \leftarrow$ LS3best($s_{new}$)
12:    end if
13:    if $f(s_{new}) < f(s_{bsf})$ then
14:        $s_{bsf} \leftarrow s_{new}$
15:        $r \leftarrow 1$
16:    else
17:        $r \leftarrow 1 + (r \bmod r_{max})$
18:    end if
19: end while
20: return $s_{bsf}$
```

– Otherwise, the shaking neighborhood size is cyclically increased w.r.t. r_{max}.

The details of local search procedures `LS1best()`, `LS2best()` and `LS3best()` are described in [17]. Shortly, `LS1best()` tries to find the best vector to move from its current partition to some other partition. Similarly, `LS2best()` finds the best simultaneous movement of two vectors, while `LS3best()` finds the best simultaneous movement of three vectors. All these procedures are executed as long as there is an improvement. We also make use of the partial (incremental) calculation of the objective function values as described in [17].

We made a small change in the re-implementation compared to the description of the original VNS. Namely, in the original work `LS1best()` was applied only when $r = 1$, while in our re-implementation `LS1best()` is always applied. This change seemed to be beneficial for the overall VNS results, because `LS1best()` is relatively cheap in terms of computational complexity, and it often provides improvements.

An additional difference is related to the application to large problem instances, which were not considered in [17]. While in [17], the authors only considered instances with up to 100 input vectors, in this work we consider much larger instances of up to 500 input vectors. In particular, using $r_{max} = 3$ turned out to be prohibitive in the case of many input vectors. This will be outlined in more detail in Section 5.1.3.

3.3. Re-implementation of the GA algorithm for MDMWNPP

The genetic algorithm making use of local search for the MDMWNPP was proposed in [2]. As the source code and/or an executable of this algorithm was not publicly available, and as the authors of the algorithm were not able to provide the source code, respectively an executable, we finally decided—as in the case of VNS—to re-implement this algorithm, which is henceforth simply labelled GA. The re-implementation was done in accordance with the algorithm description given in [2]. However, due to several ambiguities in the description provided in [2], we had to make a few implementation decisions that possibly deviate from the idea the original authors had in mind. They will be explained in the further text.

Algorithm 2 presents the general scheme of GA for the MDMWNPP. The algorithm consists of the following steps.

- First, the initial population of chromosomes (*pop*) is generated. This is done in the same way as the authors describe it in their paper, i.e. partially randomly and partially based on the problem structure. The initial population is afterwards improved by applying a local search procedure to each chromosome.
- After that, the algorithm iterates until the CPU time limit is reached. Each iteration consists of the following steps.
  - An intermediate population of offspring chromosomes (*overpop*) is generated. Hereby, the size of *overpop* is 10 times larger than the size of population *pop*.
  - The best two chromosomes from the population are directly copied to the first two positions of the intermediate population, in order to avoid losing the best solutions through the iterations. In fact, this is the first difference w.r.t. [2]. We believe this change is beneficial for the overall performance of the algorithm. The remaining chromosomes of the intermediate population are generated by the iterative use of standard genetic operators (selection, crossover, and mutation) and local search.

Algorithm 2 GA for MDMWNPP

```
1: Input: a problem instance $\mathcal{I} = (S, n, m, k)$, $pop_{size}$ – population size, $cross_{prob}$ – crossover probability, $mut_{prob}$ – mutation probability
2: Output: the best solution found within the time limit
3: $pop \leftarrow$ InitializePopulation($\mathcal{I}$, $pop_{size}$)
4: for $i = 1$ to $pop_{size}$ do
5:     $pop[i] \leftarrow$ LS($pop[i]$)
6: end for
7: Sort($pop$)
8: while time limit is not reached do
9:     $overpop_{size} \leftarrow 10 \cdot pop_{size}$
10:    $overpop \leftarrow [pop[1], pop[2]]$
11:    for $i = 2$ to $overpop_{size}/2$ do
12:        $parent1, parent2 \leftarrow$ Select($pop$),Select($pop$)
13:        $r \leftarrow U([0, 1])$
14:        if $r \leq cross_{prob}$ then
15:            $child1, child2 \leftarrow$ Crossover($parent1, parent2$)
16:        else
17:            $child1, child2 \leftarrow parent1, parent2$
18:        end if
19:        $child1, child2 \leftarrow$ Mutate($child1, mut_{prob}$), Mutate($child2, mut_{prob}$)
20:        $child1, child2 \leftarrow$ LS($child1$), LS($child2$)
21:        $overpop \leftarrow overpop + [child1, child2]$
22:    end for
23:    Sort($overpop$)
24:    $pop \leftarrow overpop[1 : pop_{size}]$
25: end while
26: return $pop[1]$
```

∗ The selection of parents is tournament-based with a tournament size of 2.

∗ Child chromosomes are formed by combining two parent chromosomes via 1-point crossover and a probability of $cross_{prob}$. In the remaining cases, no crossover is performed.

∗ Mutation is performed on both children with a per-vector probability of mut_{prob}. This means that, with a probability of mut_{prob}, a vector is moved from its current partition to another randomly chosen partition.

∗ A local search procedure (see below) is applied to each child.

    - * Both children are added to the intermediate population.
  - The intermediate population *overpop* is sorted w.r.t. the objective function value and the best pop_{size} chromosomes become the members of the next population *pop*.

The local search procedure from [2] consists of three parts:

1. The cheapest part is called *1-change neighbor*. According to the authors, this procedure is based on the random selection of a vector (a locus in the chromosome) and its move to a random partition. Its reported complexity is $O(n)$, which leads us to believe that the actual implementation is not consistent with its textual description – complexity $O(n)$ suggests that local search is exhaustive (rather than random). Therefore, instead of randomly selecting vectors, we randomly selected just its starting position (to avoid positional bias), after which all vectors from that positions onward (cyclically) were probed. Probing, as described in the paper, consists of choosing a random partition to which the vector is moved. If improvement occurs, the change is immediately applied (first-improvement strategy) and *1-change neighbor* LS is restarted.

2. If there is no further *1-change neighbor* improvement, the so-called *2-change neighbor* LS is executed. According to [2], it consists of the random selection of two vectors, after which their partitions are swapped. The complexity of this LS is $O(n^2)$. Again, this leads us to believe that the textual description is not in accordance with the actual implementation. Therefore, our re-implementation uses exhaustive probing over pairs of vectors (in a random fashion to avoid positional bias). Again, in case of an improvement, the change is immediately applied, and the *2-change neighbor* LS is restarted.

3. Last, there is also a *3-change neighbor* local search, based on the same idea as *2-change neighbor*. The complexity of this LS is $O(n^3)$.

Since all local search procedures are making a high number of objective function calls, we have implemented a fast (incremental) objective function, similar as the one used by VNS. For the experimental evaluation reported in [2], the GA parameters were set to following values: $cross_{prob} = 0.85$, $mut_{prob} = 0.1$ and $pop_{size} = 10n$. Instead of simply adopting these parameter value settings, GA is subject to parameter tuning as described in Section 5.1.

4. Adapt-Cmsa+Ls for the MDMWNPP

"Construct, Merge, Solve & Adapt" (CMSA) is a hybrid meta-heuristic originally proposed in [7]. Most of the existing applications of CMSA make an internal use of MILP solvers for solving sub-instances of the tackled problem instance. Since it was introduced, CMSA has been applied to a variety of combinatorial optimization problems (COPs) and proved to be one of the most efficient MILP-based meta-heuristics. Example applications include the one to the unbalanced minimum common string partition problem [4], the repetition-free longest common subsequence problem [5], the taxi sharing problem [3], the weighted independent domination problem [34], and the maximum happy vertices problem [14].

As mentioned above, the main idea of CMSA consists of the iterative construction of sub-instances (samples) that are further solved by means of state-of-the art (exact) solvers. Each iteration of the algorithm basically consists of four steps that can roughly be described as follows.

- The *construct* phase: a number of solutions are probabilistically generated (usually biased by an efficient greedy heuristic).
- The *merge* phase: those solution components that are (1) present in at least one of the generated solutions and (2) do not already form part of the sub-instance are added to the (initially empty) sub-instance of the tackled problem instance. Moreover, their so-called age values are set to zero.
- The *solve* phase: the updated sub-instance is solved by a solver—such as a MILP solver—under a given computation time limit.
- The *adapt* phase: this phase depends on the outcome of the solver. First, it is checked if the solution returned by the solver is a new best solution. Furthermore, those solution components that form part of the returned solution are regarded as promising by setting their age value to zero. The remaining solution components in the sub-instance are penalized by incrementing their age value. Finally, those solution components in the sub-instance whose age value has reached a limit called $age_{\max}$ are removed from the sub-instance.

As most other metaheuristics, CMSA is usually terminated after reaching a pre-defined computation time limit. As noted in [18], the original CMSA may sometimes be sensitive to the values of its control parameters. This holds especially in those cases in which the set of problem instances to be tackled is rather diverse. Therefore, as is the case for most other metaheuristics, control parameter tuning usually needs to be performed separately for different instance types. In order to alleviate this computational burden, Akbay et al. [18] recently proposed a *self-adaptive* version of CMSA, denoted by Adapt-Cmsa. This algorithm variant can be seen as a more robust version of the original CMSA. There are mainly two differences between Adapt-Cmsa and the original CMSA:

1. The parameter that controls the number of solutions generated per iteration is dynamically adjusted over time, depending on the search history.

2. Parameter $age_{\max}$ and the mechanism of sub-instance updating is replaced by a self-adaptive parameter α_{bsf} where $0 \leq \alpha_{LB} \leq \alpha_{bsf} \leq \alpha_{UB} \leq 1$. This parameter controls the balance between diversification and intensification depending on the obtained solution quality w.r.t. solution s_{bsf}, the best solution found so far. Moreover, the lower bound α_{LB} and the upper bound α_{UB} for parameter α_{bsf} are fixed. Their value is subject to parameter tuning.

In this work we adopt the general mechanism of Adapt-Cmsa from [18] in order to solve the MDMWNPP. However, as we found it necessary in the case of the MDMWNPP to improve generated solutions by applying a local search procedure, we denote our algorithm by Adapt-Cmsa+Ls. Its pseudo-code is given in Algorithm 3.

Before we describe the algorithm, remember that $s = (s_1, \ldots, s_n)$ where $1 \leq s_i \leq k$ for all $i = 1, \ldots, n$ denotes a solution to the problem. Hereby, vector i is assigned to partition s_i in solution s, for all $i = 1, \ldots, n$. In the following we call (i, s_i) a *solution component*. In this way, a solution $s = (s_1, \ldots, s_n)$ consists of the set $\{(1, s_1), \ldots, (n, s_n)\}$ of solution components. Moreover, given an instance (S, n, m, k), the set of all possible solution components is as follows:

$$C = \{(1,1), \ldots, (1,k), (2,1), \ldots, (2,k), \ldots, (n,k)\}$$

As shown in Algorithm 3, our Adapt-Cmsa+Ls algorithm starts by generating a random solution which is subsequently improved by local search. This solution is stored as the first best-so-far solution s_{bsf}. The procedure for producing a valid random solution is explained in the next section. Furthermore, the number of solutions (n_a) is set to one, the value of variable α_{bsf} is set to the upper bound α_{UB}, and the sub-instance C' is initialized by adding all solution components of s_{bsf}. Thereafter, n_a random solutions are generated by a utilizing procedure `ProbabilisticSolutionGeneration`(s_{bsf}, α_{bsf}).

Algorithm 3 Adapt-Cmsa+Ls for the MDMWNPP

```
1: Input: a problem instance $\mathcal{I} = (S, n, m, k)$ and Adapt-Cmsa+Ls parameters $t_{prop}, t_{ILP}, \alpha_{LB}, \alpha_{UB}, \alpha_{red}$
2: Output: $s_{bsf}$, the best solution found within the time limit
3: $s_{bsf} \leftarrow$ GenerateRandomSolution()
4: $s_{bsf} \leftarrow$ LS($s_{bsf}$)
5: $n_a \leftarrow 1$; $\alpha_{bsf} \leftarrow \alpha_{UB}$
6: Initialize the sub-instance $C'$ with the solution components of $s_{bsf}$
7: while CPU time limit is not reached do
8:     for $i = 1$ to $n_a$ do
9:         $s \leftarrow$ ProbabilisticSolutionGeneration($s_{bsf}, \alpha_{bsf}$)
10:        $s \leftarrow$ LS($s$)
11:        if $f(s) < f(s_{bsf})$ then $s_{bsf} \leftarrow s$ end if
12:        Add the solution components of $s$ to $C'$ (if not already in $C'$)
13:    end for
14:    $(s'_{opt}, t_{solve}) \leftarrow$ ApplyExactSolver($C', t_{ILP}$)
15:    $s'_{opt} \leftarrow$ LS($s'_{opt}$)
16:    if $t_{solve} < t_{prop} \cdot t_{ILP} \wedge \alpha_{bsf} > \alpha_{LB}$ then
17:        $\alpha_{bsf} \leftarrow \alpha_{bsf} - \alpha_{red}$ // make the subinstance harder
18:    end if
19:    if $f(s'_{opt}) < f(s_{bsf})$ then
20:        $s_{bsf} \leftarrow s'_{opt}$
21:        $n_a \leftarrow 1$
22:        $\alpha_{bsf} \leftarrow \alpha_{UB}$
23:    else if $f(s'_{opt}) > f(s_{bsf})$ then
24:        if $n_a = 1$ then
25:            $\alpha_{bsf} \leftarrow \min\{\alpha_{bsf} + \frac{\alpha_{red}}{10}, \alpha_{UB}\}$
26:        else
27:            $n_a \leftarrow 1$
28:            $\alpha_{bsf} \leftarrow \alpha_{UB}$
29:        end if
30:    else
31:        $n_a \leftarrow n_a + 1$
32:    end if
33:    Re-initialize the sub-instance $C'$ with the solution components of $s_{bsf}$
34: end while
35: return $s_{bsf}$
```

The generation of solutions at each iteration is biased by α_{bsf}: the larger the value of α_{bsf}, the larger will be the similarity between a (probabilistically) generated solution and s_{bsf}, and vice-versa. Each generated solution is improved by means of the `LS`() procedure. After that, all solution components from these solutions are added to C' (if not already there). Moreover, s_{bsf} is updated in case a new best-so-far solution is found.

Next, the current sub-instance C' is (approximately) solved in function `ApplyExactSolver`(C', t_{ILP}) as follows. The MILP solver CPLEX is applied to a restricted version of the Coam model from Section 3. This restricted MILP model is obtained by adding the following constraints to Coam:

$$x_{c',c''} = 0, \forall c = (c', c'') \notin C'$$

This restricted model w.r.t. the set of solution components present in C' will be denoted by Coam(C') in the further text. For example, if a sub-instance C', concerning some vector v_i, contains the following solutions components, $\{(i, j), (i, k)\}$, it means that we only allow placing vector v_i in partition j or in partition k. Note that CPLEX is given a computation time limit of t_{ILP} CPU seconds for solving the restricted MILP at each iteration. Additionally, the time needed to solve the sub-instance is monitored and stored in t_{solve}. Afterwards, the solution produced by the solver is possibly improved by applying the `LS`() procedure.

Next, parameter α_{bsf} value is adapted w.r.t. (1) the computation time t_{solve} required to solve the present sub-instance and (2) the quality of the obtained solution s'_{opt}. Namely, if t_{solve} is low—that is, within a ratio t_{prop} of the allowed solver time t_{ILP}—the sub-instance is considered to be easy to solve. Therefore, a stronger diversification is needed for the probabilistic construction of solutions in function `ProbabilisticSolutionGeneration`(s_{bsf}, α_{bsf}). This is achieved by decreasing α_{bsf} by a value α_{red}. Of course, in case $\alpha_{bsf} \leq \alpha_{LB}$, where α_{LB} is the lower bound for α_{bsf}, α_{bsf} remains unchanged. In those cases in which s'_{opt} is better than s_{bsf}, s_{bsf} is updated accordingly, n_a is set to 1, while $\alpha_{bsf} = \alpha_{UB}$. This implies that exploration will be performed in the close neighborhood of the new best-so-far solution within the next iteration (increased intensification). On the other hand, in case s'_{opt} and s_{bsf} are equal, we increment n_a, which is done for the following reason. By increasing the number of probabilistically generated solutions that enter the sub-instance, the sub-instance will be larger and the search will be given a higher chance to improve over s_{bsf} (increased diversification). Finally, in the last remaining case—that is, if s'_{opt} is worse than s_{bsf}, the intensification around s_{bsf} should be increased. There are two ways for making intensification stronger under these circumstances. Firstly, when $n_a = 1$, i.e. the number of random solutions around s_{bsf} is already minimal, intensification is increased by slightly increasing α_{bsf}, which has the effect that the probabilistically generated solutions become more similar to s_{bsf}. Otherwise, if $n_a > 1$, we reset n_a to one and we set $\alpha_{bsf} = \alpha_{UB}$. In this way, intensification will be increased maximally in order to fully focus the search on the close neighborhood around s_{bsf}.

Note that at the end of a major iteration of Adapt-Cmsa+Ls, C' is reinitialized by the solution components of the best-so-far solution s_{bsf}. This concludes the description of the main algorithm. In the remainder of this section the functions for generating a random solution, for probabilistically generating a solution with a bias towards s_{bsf}, and for local search are described in detail.

Function `GenerateRandomSolution`(): First, k vectors are chosen uniformly at random from S. Subsequently they assigned them to different partitions. In this way, solution feasibility can be assured at the end of the solution generation. In a second step, the remaining vectors—that are not yet assigned to partitions—are assigned to random partitions.

Function `ProbabilisticSolutionGeneration`(s_{bsf}, α_{bsf}). The core idea of this method is to construct random solutions whose structural closeness to s_{bsf} is controlled by means of parameter α_{bsf}. More precisely, as α_{bsf} gets closer to 0, the similarity between solutions generated by the procedure and s_{bsf} decreases. On the contrary, as the value of α_{bsf} gets closer to 1, the generated solutions become structurally more and more similar to s_{bsf}. For example, if $\alpha_{bsf} = 1$, the method will simply return solution s_{bsf}. However, this specific case is not of particular interest. The method that we designed is given in Algorithm 4. In particular, this method applies a mutation scheme as known from the field of Evolutionary Computation [6]. Each vector is assigned a probability of $1 - \alpha_{bsf}$ to be relocated to a randomly chosen partition (including the one it is currently assigned to). The obtained solution is then rearranged according to the rule of symmetry breaking outlined in Section 2.

Function `LS`(s): during the development of our algorithm we realized that the application of local search (LS) for improving the generated solutions significantly improved the performance of the algorithm. As already mentioned before, LS is called in two occasions. First, LS is applied both after the generation of the initial solution and after the probabilistic generation of solutions at each iteration. Second, LS is applied to the solution returned

Algorithm 4 Function `ProbabilisticSolutionGeneration`(s_{bsf}, α_{bsf})

```
1: Input: $s_{bsf}, \alpha_{bsf}$
2: Output: a mutated solution $s'_{bsf}$
3: $s'_{bsf} \leftarrow s_{bsf}$
4: for $i = 1$ to $n$ do
5:     if the "$s'_{bsf}[i]$"-th partition contains at least two vectors then
6:         $r \leftarrow U([0,1])$
7:         if $r \leq 1 - \alpha_{bsf}$ then
8:             $pos \leftarrow U(\{1,\ldots,k\})$
9:             $s'_{bsf}[i] \leftarrow pos$
10:        end if
11:    end if
12: end for
```

by the exact solver (note that when the exact solver verifies optimality, local search is redundant). Calling LS after the generation of a solution reduces the diversity of solution components, which results in a stronger intensification than when not using LS at all. On the other hand, applying LS to the output of the exact solver does not change the intensification/diversification balance of the algorithm. It rather enhances the solving phase in those scenarios in which the sub-instance is not solved to optimality. The LS pseudo-code is given in Algorithm 5. It consists of two parts:

- In the first phase—see lines 7–22—local search based on the *1-move* strategy is performed until no further improvements can be found in this way. Note that this neighborhood is explored in a *best-improvement* way.
- The second phase—see lines 23–36—consists in swapping those two vectors from different partitions that results in the best improvement possible. In other words, exactly one move in the *2-swap* neighborhood (best-improvement) is performed. In case of success, the LS procedure switches again to the first phase.

It is important to note that—in all occasions—a fast (incremental) calculation of the objective function is performed. After the application of LS, the obtained local optimum is then rearranged according to the rule of symmetry breaking (see Section 2).

Algorithm 5 Function `LS`(s)

```
1: **Input**: solution $s$
2: **Output**: possibly improved solution $s_{LS}$
3: $s_{LS} \leftarrow s$
4: $best_{fit} \leftarrow f(s)$
5: $improved \leftarrow True$
6: while $improved$ do
7:     while $improved$ do
8:         $improved \leftarrow False$
9:         for $i = 1$ to $n$ do// 1-move LS with the best-improvement strategy
10:            for $p = 1$ to $k$ do
11:                if $p$ allowed then // moving to $p$ produces feasible solution
12:                    $s_{LS}[i] \leftarrow p$
13:                    if $f(s_{LS}) < best_{fit}$ then
14:                        $best_{fit} \leftarrow f(s_{LS})$
15:                        $best_i \leftarrow i, best_p \leftarrow p, improved \leftarrow True$
16:                    end if
17:                    $s_{LS}[i] \leftarrow s[i]$
18:                end if
19:            end for
20:        end for
21:        if $improved$ then $s[best_i] \leftarrow best_p$ end if // continue 1-move LS
22:    end while
23:    for $i = 1$ to $n - 1$ do
24:        for $j = i + 1$ to $n$ do
25:            $p_i \leftarrow s_{LS}[i], s_{LS}[i] \leftarrow s_{LS}[j], s_{LS}[j] \leftarrow p_i$ // swapping partitions
26:            if $f(s_{LS}) < best_{fit}$ then
27:                $best_{fit} \leftarrow f(s_{LS})$
28:                $best_i \leftarrow i, best_j \leftarrow j$
29:                $improved \leftarrow True$
30:            end if
31:            $p_i \leftarrow s_{LS}[i], s_{LS}[i] \leftarrow s_{LS}[j], s_{LS}[j] \leftarrow p_i$ // swapping back
32:        end for
33:    end for
34:    if $improved$ then
35:        $p_i \leftarrow s[best_i], s[best_i] \leftarrow s[best_j], s[best_j] \leftarrow p_i$
36:    end if
37: end while
38: return $s_{LS}$
```

5. Experimental Evaluation

The following approaches for solving the MDMWNPP are included in the experimental comparison presented in this section:

- CPLEX applied to the MILP model from Faria et al. [16], labeled Faria;
- CPLEX applied to the MILP model from Nikolic et al. [26], labeled Coam;
- Our re-implementation of the VNS approach from Faria et al. [17], as explained in Section 3.2 and labelled Vns;
- Our re-implementation of the GA algorithm from [2] as explained in Section 3.3 and labeled Ga;
- Our Adapt-Cmsa+Ls approach, as presented in Section 4.

All methods were implemented in C++. Moreover, GCC 9.4 was used for compiling the software. Experiments were conducted on hardware with an Intel Xeon E5-2640 CPU with 2.40 GHz and 8 GB of RAM. Note also that all experiments were conducted in single-threaded mode (also the ones invovling CPLEX). Standalone MILP models and restricted MILP models within CMSA were solved by Cplex [24] (version 12.1). Standalone MILP models were solved exactly once, while the stochastic methods were applied 10 times to each problem instance.

Benchmark instances. All problem instances are generated from five basic data files named `mdmwnpp_500_20x.txt`, where $\texttt{x} \in \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\}$ [12]. Each of these files contains 500 decimal random vectors $v_i = (v_{i,1}, \ldots, v_{i,20})$, $i = 1, \ldots, 500$. For example, the specific problem instance with $n = 100, m = 10, k = 2$ of type `A` is obtained from file `mdmwnpp_500_20a.txt` by extracting the first 100 vectors $v_1, \ldots, v_{100}$ and the first $m = 10$ coordinates of these vectors.

From each of these five basic data files, 48 specific instances were generated. This was done by combining the following values of n, m and k $(3 \times 4 \times 4 = 48)$:

- $n \in \{50, 100, 500\}$
- $m \in \{2, 5, 10, 20\}$
- $k \in \{2, 5, 10, 20\}$

That is, $48 \times 5 = 240$ specific problem instances were used in total for our experimental evaluation.

Moreover, both Ga and Vns were additionally evaluated on the set of instances obtained by combinations of $n \in \{50, 100\}$, $m \in \{2, 3, 4, 5, 10, 15, 20\}$, and $k \in \{3, 4\}$. Thus, this set of $2 \times 7 \times 2 = 28$ additional instances (only for benchmark set A, as reported in [2, 17]) was also used in our experimental evaluation.

5.1. Parameter tuning

In this section, we give the details on the tuning process of the following three algorithms: our two re-implementations, that is Vns and Ga, and Adapt–Cmsa+Ls. Concerning Vns and Ga, tuning is performed for instances with $k \geq 5$, while for the remaining instances ($k \in \{2, 3, 4\}$) the algorithms' configurations are taken from the literature, if not indicated differently. For the purpose of tuning, we used the well-known `irace` package [19], a tool capable of automatically configuring algorithms on the basis of in-advance given domains of the parameters. This tool is implemented in programming language `R` and utilizes the iterated racing procedure, an extension of the iterated F–race method [35] to search for the best algorithm configurations w.r.t. given parameter domains and tuning instances. As all five basic data files (see above) consist of randomly generated real vectors, we decided to tune our algorithms on the basis of type A instances.

5.1.1. Tuning of the parameters of Adapt-Cmsa+Ls

For the purpose of tuning the parameters of Adapt-Cmsa+Ls, the parameter domains as specified in Table 1 are used.

Table 1: Domains of the Adapt-Cmsa+Ls parameters

t_{ILP}	α_{LB}	α_{UB}	α_{red}	t_{prop}
$\{1, 2, \ldots, 99, 100\}$	$[0.2, 0.8]$	$[0.85, 0.98]$	$[0.03, 0.3]$	$[0.05, 0.6]$

Three separate `irace` runs were executed, one for each value of $n \in \{50, 100, 500\}$. In this way, tuning is performed for small, medium, and large-sized instances. Each `irace` run was based on nine sample instances derived from file `mdmwnpp_500_20a.txt`, for all combinations of values $m \in \{5, 10, 20\}$ and $k \in \{5, 10, 20\}$. A budget of 1000 algorithm executions was given to each `irace` execution. The best configurations for Adapt-Cmsa+Ls obtained by `irace` are displayed in Table 2.

Table 2: Obtained parameter configurations for Adapt-Cmsa+Ls

n	t_{ILP}	α_{LB}	α_{UB}	α_{red}	t_{prop}
50	2	0.518	0.898	0.093	0.557
100	2	0.302	0.961	0.049	0.264
500	6	0.302	0.948	0.239	0.069

5.1.2. Tuning of the parameters of Ga

In order to tune the parameters of our re-implementation of Ga, the parameter domains as shown in Table 3 were used.

Table 3: Domains of Ga algorithm parameters

$pop_{size}(\times n)$	$cross_{rate}$	mut_{rate}
$\{0.5, 1.0, 1.5, 2.0, \ldots, 4.5, 5.0\}$	$[0.05, 0.95]$	$[0.05, 0.3]$

Again, three separate `irace` runs were executed, one per each value of $n \in \{50, 100, 500\}$, referring to tuning the algorithm on small, medium, and large-sized instances. Each `irace` run was allowed a budget of 1000 algorithm executions. The same set of test instances were used for tuning as in the case of Adapt-Cmsa+Ls. The obtained configurations for the Ga algorithm are presented in Table 4.

Table 4: Obtained parameter configurations for Ga

n	pop_{size}	$cross_{rate}$	mut_{rate}
50	1.0	0.891	0.234
100	2.0	0.784	0.168
500	1.0	0.343	0.237

5.1.3. Tuning of the parameters of Vns

In the case of the Vns algorithm just one parameter has to be tuned, that is $r_{\max}$. As already emphasized, due to a high complexity of the second and third neighborhood utilized by Vns, when applied to instances with a high number of vectors—that is, $n = 500$—it turned out to be very inefficient to use any but the first neighborhood. Thus, in the case of instances with $n = 500$, we set $r_{\max} = 1$. Moreover, during preliminary experiments we noticed that the value of k had the highest influence on the sensibility and efficiency of the algorithm. According to that, all combinations of $n \in \{50, 100\}$, and $m \in \{5, 10, 20\}$ were used to derive six training instances from file `mdmwnpp_500_20a.txt`, for each $k \in \{5, 10, 20\}$. Subsequently, three separate `irace` runs were executed, one for each value of k, and each using a budget of 1000 algorithm executions. Summarizing, the best configurations of Vns obtained from irace are as follows:

- if $n = 500$, then $r_{\max} = 1$;
- if $k = 5$, and $n \in \{50, 100\}$ then $r_{\max} = 1$;
- if $k = 10$, and $n \in \{50, 100\}$, then $r_{\max} = 2$;
- if $k = 20$, and $n \in \{50, 100\}$ then $r_{\max} = 1$.

5.2. Algorithm comparison

In this section we comprehensively compare the results of five different approaches. First, we compare our algorithm and the re-implementations of Ga and Vns to the original results of the two latter algorithms as reported in the literature, taking care of being as fair as possible with the comparisons. Thereafter, for the remaining set of instances—for which no results of Ga and Vns are available in the literature—we evaluate our re-implementations of the two algorithms with our Adapt-Cmsa+Ls approach and with the application of Cplex to the two available MILP models. Finally, we also provide a statistical evaluation of the results obtained by the five approaches (in addition to the know results from the literature, where applicable).

However, before starting with the comparison across different values of k, we briefly show an overall head-to-head comparison of the original and the re-implemented versions of GA and VNS. Note that the original algorithm versions are henceforth labelled Ga-Lit and Vns-Lit. As can be seen from Table 5, the Ga from literature is better when it comes to the number of best solutions (27/40 versus 13/40), while the re-implemented Ga is better concerning the average solution quality (28417.28 versus 31466.58). Differences between the original and the re-implemented VNS are less obvious. In fact, they are not significant as we will confirm later by means of statistical tests.

5.2.1. Comparison in case of instances with $k = 2$

Numerical results for the set of instances with $k = 2$ concerning data file (benchmark set) `A` are provided in Table 6. The results for the remaining benchmark sets—that is `B, C, D` and `E`—are provided in Appendix A. In addition to the results produced by ourselves, the table(s) also contain the results of the original Ga approach (Ga-Lit) as published in [2]. The first three columns of the table(s) show the instance characteristics. The following

Table 5: Head-to-head comparison between the original and the re-implemented versions of GA and VNS on the datasets from the literature.

	avg. sol. quality	**# times better**
Ga	28417.28	13/40
Ga-Lit	31466.58	27/40
Vns	36665.46	9/28
Vns-Lit	35087.72	19/28

Table 6: Comparison with the related literature for instances with $k = 2$ (benchmark set A).

n	m	k	Faria	Coam	Vns	Ga	Adapt-Cmsa+Ls	Ga- Lit	Lit - best
50	2	2	2.625	0.875	42.90	1.12	2.51	5.438	**0.48**^{b}
50	5	2	**920.125**	926.250	6763.29	1084.88	2456.40	925.226	920.32^a
50	10	2	16173.500	16173.500	36832.34	16365.51	18196.61	**15722.290**	16175.67^b
50	20	2	52826.375	52826.375	86204.41	54863.73	56692.47	**50647.836**	56919.47^b
100	2	2	13.750	19.250	38.48	19.19	24.00	19.513	**0.62**^{a}
100	5	2	2453.750	2447.000	6422.44	3267.70	3501.39	**2347.346**	2410.86^a
100	10	2	16010.500	15998.000	35671.53	17082.52	19599.97	**15792.742**	15990.02^a
100	20	2	48152.750	55236.500	86973.64	51772.25	55217.38	**46782.412**	52436.03^b
500	2	2	13.000	19.000	34.35	9.71	4.76	1.354	**0.12**^{a}
500	5	2	3526.000	3685.000	7251.19	634.58	2298.82	1922.135	**2.44**^{a}
500	10	2	21108.000	20071.000	40458.64	15653.14	18816.20	12938.304	**12896.61**^{a}
500	20	2	49877.000	54547.000	132146.10	51244.55	53518.80	**32934.495**	42461.60^b
average:			17589.781250	18495.812500	36569.942500	17666.573333	19194.109167	**15003.257583**	16684.52
#best:			1	0	0	0	0	**6**	5

columns provide the results of Faria, Coam, Vns, Ga, Adapt-Cmsa+Ls and Ga-Lit, respectively. The last column reports the best results from [31] where three specialized algorithms for the case $k = 2$ are compared in the context of average solutions quality: iMADEB, MADEB from [31, 32], and GRASP with path re-linking from [37]. Each of the results in the last column is marked by a superscript letter denoting which of the three algorithms achieved this result: iMADEB in case of superscript a, MADEB in case of superscript b. Note that the GRASP algorithm with path relinking did not achieve any of these results. Finally, the table contains two (last) rows indicating the overall average results and the number of instances for which the best solution has been found by each of the considered algorithms.

As the purpose of these experiments includes checking the efficiency of our re-implementation of Ga w.r.t. Ga–Lit, we made sure of establishing fair conditions for this comparison. The experiments reported in [2] were conducted on a machine equipped with a processor of 1.6 GHz and the average runtime over all experiments was approx. 900s per experiment. As our machine is equipped with a processor of 2.4 GHz, we decided to fix the runtime of all five competitors to $900 \times 1.6/2.4 = 600$ CPU seconds. Note that in [31] the authors used the same computation time limit, tested under a very similar processor configuration as ours. Moreover, our Vns approach used the same parameter configurations as reported in [17]. The configurations of our Ga are set according to Table 4, while the configurations of Adapt-Cmsa+Ls are set according to Table 2.

The following conclusions may be drawn from the obtained experimental results for instances with $k = 2$ from benchmark set A:

- The best performing algorithm in terms of average solution quality is Ga–Lit (results reported in [2]). Ga-Lit is followed by the combined results of the specialized algorithms for $k = 2$ from [31], achieved by iMADEB [31] and the basic MADEB algorithm [32]. The third-best approach is Faria followed by our re-implementation of Ga, Coam and Adapt–Cmsa+Ls, respectively. The worst-performing approach in terms of delivered average solution quality is clearly the re-implementation of Vns.
- On six (out of twelve) instances Ga-Lit was able to obtain the best average solution quality, while the remaining five best results were obtained by the specialized algorithms for $k = 2$ (Lit - Best). Faria was able to achieve a best solution in just one instance, while no other algorithms were able to achieve a best solution for any instance.
- Similar conclusions can be drawn in the case of the remaining benchmark sets (B, C, D and E); see Appendix A and the respective tables.
- Note that the results of Ga-Lit are better than those of our re-implemented Ga which indicates that either there are some ambiguities in the description of Ga as given in [2], or that certain aspects were implemented with different data structures, for example. However, as pointed out before, we did everything in order to achieve the best re-implementation of this algorithm as possible. Moreover, as we will see in the next section, our re-implementation scales much better with an increasing value of k (number of partitions) in comparison to the results reported in literature.

5.2.2. Comparisons to the literature results for instances with $k \in \{3, 4\}$

In this section we compare our results in the context of instances with $k \in \{3, 4\}$ from benchmark set A that were used in [2, 17]. Table 7 presents numerical results of the five approaches as well as the results of the original Vns and Ga approaches from the literature. Table 7 has basically the same structure as Table 6 from the previous section, with two exceptions. First, there is no column Lit-best (as Lit-best referred to results from algorithms specialized for $k = 2$). Second, an additional column (with heading VNS-Lit) reports on the results of the original VNS from [17].

Concerning the computation time limits used to execute the five approaches that we implemented ourselfs, they are chosen in order to be fair with the original execution of GA-Lit in [2]. Namely, the average computation time of GA-Lit over all instances reported in the aforementioned paper is around 1800 CPU seconds. Due to the afore-mentioned difference in the CPU processor speeds of the utilized machines (1.6GHz vs. 2.4GHz), we decided to use a computation time limit of 1200 CPU seconds for all five approaches. We believe that, in this way, none of the approaches enjoyed any significant advantage over the others.

Concerning the parameter configurations of our re-implementation of VNS, we make use of all three neighborhoods—that is, $r_{\max} = 3$—as in [17]. Furthermore, for GA we used the same parameter configuration as reported for the original GA in [2]. This was done in order to achieve a fair comparison of our re-implementations of GA and VNS and their original versions. For ADAPT-CMSA+LS approach we used the parameter configuration from Table 1.

The following conclusions may be drawn from the obtained numerical results:

- In terms of average solution quality, ADAPT-CMSA+LS is clearly the best algorithm. The second-best approach is our re-implementation of GA, being slightly better than COAM and FARIA. The two worst-performing approaches are VNS followed by GA-Lit.

- The results of our implementation of GA are significantly better than those of GA-Lit as reported in [2]. It seems that our re-implemented GA scales better with an increasing problem instance size, and especially with an increase of k (number of partitions).

- Concerning our re-implemented VNS and the results of VNS-Lit as reported in [17], we tested the statistical significance of the results by means of the Wilcoxon ranked sum test. The result of the test was that there are no statistical differences between the results of these two implementations, although a slight advantage in the average solution quality in a favor of VNS-Lit can be noticed.

- Concerning number of instances for which best solutions were found, the best approach is ADAPT-CMSA+LS, being successful in 13 (out of

Table 7: Comparison for instances with $k \in \{3, 4\}$ (those instances from benchmark set `A` that were used in [2, 17]).

n	m	k	Faria	Coam	Vns	Ga	Adapt-Cmsa+Ls	Ga - Lit	Vns - Lit
50	2	3	317.9375	311.1250	177.15	164.00	**29.19**	86.2	130.98
50	2	4	846.8125	294.0625	849.66	672.34	**132.65**	391.4	321.69
50	3	3	1405.3750	1387.8750	1768.05	1544.96	491.00	**334.4**	1045.94
50	3	4	4202.6250	5623.4375	4887.44	3776.72	1634.93	**678.3**	687.75
50	4	3	5492.6250	5569.4375	5393.63	4919.05	2178.63	3382.5	**2044.01**
50	4	4	9197.5000	9430.5625	11214.58	9833.64	5307.93	**836.7**	3185.66
50	5	3	10308.8125	10555.3750	11155.83	9386.20	5014.52	**4125.8**	14266.40
50	5	4	21140.6250	15535.0000	21044.33	16480.32	11268.64	**1094.4**	19979.40
50	10	3	**28724.6875**	40155.2500	44297.50	36934.34	29282.76	37521.6	38136.10
50	10	4	50224.6250	50849.7500	57326.62	49545.68	45498.90	**42005.6**	57025.20
50	15	3	63736.6875	61752.1875	73299.99	63701.58	58045.49	**56015.2**	68396.10
50	15	4	85889.3750	79702.4375	87108.10	77555.69	75205.67	**56034.7**	91035.60
50	20	3	85887.1875	**81206.3750**	95596.65	84929.06	81333.47	102652.0	92299.10
50	20	4	101370.6250	107893.3750	107167.42	103264.78	**97517.29**	123627.8	108404.00
100	2	3	193.2500	**30.2500**	134.92	141.39	38.20	178.1	50.70
100	2	4	601.8750	935.2500	545.64	570.57	**51.84**	687.2	99.28
100	3	3	1249.6250	1208.3750	1727.35	1302.33	683.28	**531.3**	2886.44
100	3	4	4938.8750	3709.1250	4096.42	2831.90	**779.76**	1213.7	5765.02
100	4	3	4529.1250	4356.8750	5220.42	4212.97	2257.03	**867.5**	8374.57
100	4	4	10313.8750	9903.5000	10400.96	7536.92	3023.79	**1924.6**	12219.90
100	5	3	10418.1250	9686.6250	11654.32	8929.78	**4487.51**	6224.5	11527.30
100	5	4	13800.3750	13307.7500	16714.95	14108.68	**6514.77**	8356.8	19056.70
100	10	3	33188.6250	39677.6250	41304.08	33495.31	**28523.88**	47004.8	37146.20
100	10	4	45014.0000	58699.3750	55965.44	45761.70	**33450.13**	75034.8	58992.60
100	15	3	59056.5000	60969.1250	69674.96	61119.63	**54162.51**	96827.3	61427.60
100	15	4	91919.3750	78554.0000	85095.32	75902.54	**65026.44**	122892.7	76239.60
100	20	3	95286.3750	86192.0000	91248.08	83530.70	**75659.66**	113112.5	84093.40
100	20	4	108824.2500	108505.3750	111563.15	100815.74	**85404.72**	174981.6	107619.00
average:			33859.991071	33785.767857	36665.462857	33024.730714	**27607.306786**	38522.285714	35087.722857
#best:			1	2	0	0	**13**	11	0

28) cases. The second-best is Ga-Lit, which is successful in 11 cases. Note that Ga-Lit performs strongly in the case of small–sized instances (with low values of n and/or k), while Adapt-Cmsa+Ls performs significantly better when it comes to larger instances (especially for $n = 100$). This indicates that Adapt-Cmsa+Ls scales better than the other approaches. In fact, this will be confirmed in the next section.

5.2.3. Comparison for instances with $k \geq 5$

Numerical results for instances from all benchmark sets (from `A` to `E`) are provided in Tables 8-12. In particular, the shown results are averaged over 4 instances (with varying values of m) and 10 algorithm runs per instance (apart from Coam and Faria, which were applied exactly once to each problem instance). After the first two table columns indicating n and k, each table contains the following columns: the third and fourth columns provide the average solution quality ($\overline{obj}$) of Faria and Coam, respectively, The next three blocks of columns report on the solution qualities achieved by Vns, Ga, and Adapt-Cmsa+Ls, respectively. Hereby, each block contains a column for the average solution quality $\overline{obj}$ (over 10 runs per instance) and a column for the qualities of the best-found solutions ($\overline{obj}_{best}$) averaged over four instances. Best average results are shown in bold font. Averages over the columns are shown in the second row from the bottom, while the last row af each table provides the number of instances for which the best solution has been found by the respective algorithm. The CPU time limit for all algorithms and runs was set to 1800 seconds.

Table 8: Numerical results for instances of set `A`.

		Faria	Coam	Vns		Ga		Adapt-Cmsa+Ls	
n	k	$\overline{obj}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$
50	5	52480.273438	50448.406250	43602.53825	48260.579500	46678.37900	48739.040450	37975.22225	**42676.246175**
50	10	74552.242188	74043.105469	59160.21125	63609.510825	60741.85950	63050.489150	54153.73200	**56512.690300**
50	20	97281.359375	88614.673828	83437.30425	91552.631150	76159.97625	77935.876125	73211.43750	**74249.380700**
100	5	52543.984375	52656.359375	43008.36400	45978.663050	45063.45075	46971.570925	33547.96075	**35386.264600**
100	10	75361.281250	72256.781250	61684.26375	67744.604400	58763.28150	66972.767650	50489.01175	**52208.099000**
100	20	114458.984375	94477.742188	77709.14550	84318.964625	73825.90700	81477.332575	61855.09200	**63432.350175**
500	5	49724.875000	52751.875000	40178.27150	47653.834675	38348.78450	42735.434425	32456.36175	**33897.967300**
500	10	82210.875000	67769.125000	56624.35675	63557.650575	50156.61650	54411.170225	44612.26800	**45088.109400**
500	20	126289.531250	95385.937500	102938.37425	120360.418075	58148.74075	66397.871175	55120.69800	**56035.950725**
average:		80544.822917	72044.889540		70337.428542		60965.728078		**51054.117597**
#**best**:		0	0		0		0		**36**

Table 9: Numerical results for instances of set `B`.

		Faria	Coam	Vns		Ga		Adapt-Cmsa+Ls	
n	k	$\overline{obj}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$
50	5	51177.765625	55740.101562	46453.833500	50329.964275	46423.910500	50084.186075	37975.222250	**42676.246175**
50	10	75443.601562	80852.105469	60120.504500	64585.905725	62380.194750	65193.646875	54153.732000	**56512.690300**
50	20	103236.794922	90294.685547	89501.946750	95170.036850	81286.982250	82822.785500	73211.437500	**74249.380700**
100	5	55555.859375	51131.734375	42619.043250	46259.218400	45430.129750	48064.432175	33547.960750	**35386.264600**
100	10	82270.023438	72310.289062	64275.776250	68331.355450	59935.975000	69128.336075	50489.011750	**52208.099000**
100	20	107080.718750	99085.972656	78474.268500	85616.907700	75304.677250	85937.622550	61855.092000	**63432.350175**
500	5	69325.000000	65162.000000	53732.823000	62440.645367	54341.332667	59785.343000	43234.032333	**45119.943600**
500	10	107032.083333	97416.750000	77988.785667	86855.548900	65238.388000	72466.533967	59298.372000	**59887.724767**
500	20	162537.875000	125619.875000	145754.264000	162031.269800	83559.443667	91409.661267	73054.540333	**74262.867400**
average:		88355.876657	80674.406960		78035.028661		68966.928839		**55626.416094**
#**best**:		0	0		0		0		**36**

Table 10: Numerical results for instances of set `C`.

		Faria	Coam	Vns		Ga		Adapt-Cmsa+Ls	
n	k	$\overline{obj}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$
50	5	57580.531250	55551.820312	46567.84750	51222.114300	48570.73700	51302.244975	41530.09525	**43989.046275**
50	10	80534.976562	79257.027344	62736.28875	67872.648600	62959.32325	65386.458425	59818.55725	**61084.495775**
50	20	106602.119141	90731.488281	90106.75925	96006.631275	79301.19900	81694.241525	75964.39500	**78537.226600**
100	5	55010.265625	51036.765625	40195.57850	45236.703450	43040.55875	46665.278950	36113.23650	**38117.935750**
100	10	78355.718750	77978.054688	62783.43050	69294.168450	63311.43850	69131.969725	52152.85575	**53583.505175**
100	20	107232.851562	98042.050781	80558.14600	86990.708650	72118.42800	83022.812475	62199.65175	**63818.361900**
500	5	51580.625000	51523.500000	41886.59000	47489.250900	38325.51925	44525.426000	34017.74200	**34423.386750**
500	10	78917.812500	76269.437500	57309.51375	63868.418175	52048.27600	55150.334600	44770.79700	**45887.724200**
500	20	133680.500000	100940.812500	104692.93050	118724.632250	58772.68400	71862.630650	57250.37100	**57783.954375**
average:		83277.266710	75703.439670		71856.141783		63193.488592		**53025.070756**
#**best**:		0	0		0		0		**36**

Table 11: Numerical results for instances of set D.

		Faria	Coam	Vns		Ga		Adapt-Cmsa+Ls	
n	k	$\overline{obj}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$
50	5	52722.523438	52511.054688	46438.21775	50119.441175	47721.01075	50317.955150	41723.54200	**44115.056800**
50	10	76588.000000	77985.046875	62007.21025	65902.012950	62168.48650	65633.364025	58831.10100	**60954.307375**
50	20	103887.498047	95599.542969	87363.97050	94905.408075	79230.26400	82028.862975	78114.77425	**79720.241875**
100	5	55021.484375	49089.796875	39157.51175	44677.900775	43746.70050	46612.200350	32487.27650	**35013.209475**
100	10	77907.398438	77904.054688	61937.34725	67131.737075	59536.78175	68816.347800	51174.78750	**52657.089625**
100	20	109512.062500	91041.835938	79754.95125	85740.222325	71585.53400	80702.630925	61058.89400	**63027.354450**
500	5	53325.875000	51141.375000	40982.49525	46176.690075	42167.07850	45964.860425	32317.96375	**33496.323850**
500	10	79939.375000	76752.062500	58264.81550	64047.749850	47837.12725	54195.767000	45894.50450	**46399.119550**
500	20	115068.125000	93760.843750	106092.04800	122768.206700	57828.50925	65759.571900	57551.79275	**57762.288550**
average:		80441.371311	73976.179253		71274.374333		62225.728950		**52571.665728**
#best:		0	0		0		0		**36**

Table 12: Numerical results for instances of set E.

		Faria	Coam	Vns		Ga		Adapt-Cmsa+Ls	
n	k	$\overline{obj}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$	$\overline{obj}_{best}$	$\overline{obj}$
50	5	59180.671875	51808.585938	46535.47325	50524.878775	49364.62000	51800.529875	42139.70900	**46207.731500**
50	10	83591.144531	78656.062500	62542.98575	67378.651850	64306.78300	66917.611775	57730.75900	**60711.596550**
50	20	110378.949219	93354.359375	92320.40525	98133.122400	83569.62500	85775.873925	80057.10000	**81883.902700**
100	5	52455.734375	55958.453125	43136.01050	46203.926500	44086.84425	47736.317250	34147.98950	**37063.122850**
100	10	81878.367188	75980.765625	64232.72750	68700.418150	63148.40000	72326.877400	52257.74775	**53902.971075**
100	20	105608.503906	98227.203125	80086.90800	86557.766575	72441.85450	84939.003550	63179.15250	**65171.232300**
500	5	52203.625000	48451.625000	39112.68925	46918.158825	42141.17750	45971.129350	33670.37850	**34001.313400**
500	10	79189.437500	71592.562500	57052.13325	64458.622950	50035.10225	55385.909075	45888.24775	**46413.464900**
500	20	140641.812500	102363.562500	106337.21775	120260.400575	61653.69850	70682.052550	55549.67975	**58505.280025**
average:		85014.249566	75154.797743		72126.216289		64615.033861		**53762.290589**
#best:		0	0		0		0		**36**

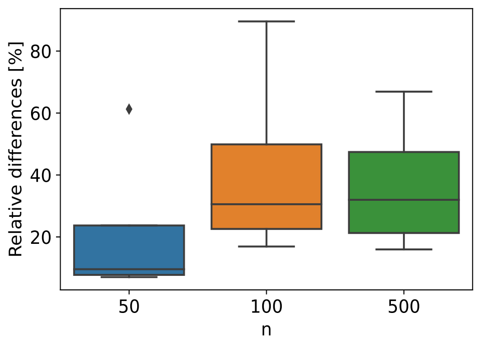
The following conclusions may be drawn from the obtained experimental results:

- The best average results are produced—in all cases; thus in 36 (out of 36) cases—by Adapt-Cmsa+Ls. In our opinion, this is due, especially in the case of these rather large data sets, to the interplay between heuristic and exact algorithm elements that guides the search towards more promising subinstances, for which high-quality solutions can be obtained by the solver within rather low computation times.

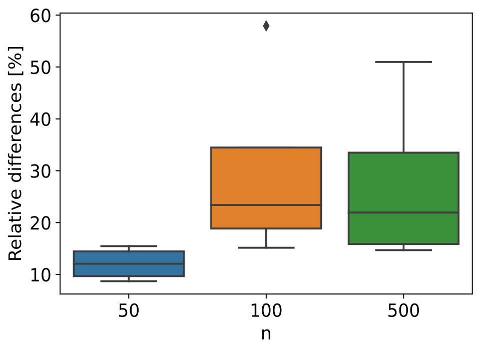
- The Adapt-Cmsa+Ls approach delivers significantly better results than the other four approaches. This is also indicated by the plots shown in Figure 1. They show the relative differences between the results of Adapt-Cmsa+Ls and the combined best results of the other (four) approaches, on benchmark set A. There are three plots, showing the results for instances grouped by $k \in \{5, 10, 20\}$. Furthermore, each plot contains three box plots, one for each considered value of n (x axis). They show the relative differences for the results on respective instance sub-groups. The relative average differences range from $\approx 5\%$ in the case of the small-sized instances, to $\approx 30\%$ in case of the larger instances in favor of Adapt-Cmsa+Ls. The same plots are also provided in the context of the other four benchmark sets; see Appendix B.

- While Vns and Ga perform equally good in the case of instances with small n or k, Ga performs much better than Vns for the remaining cases.

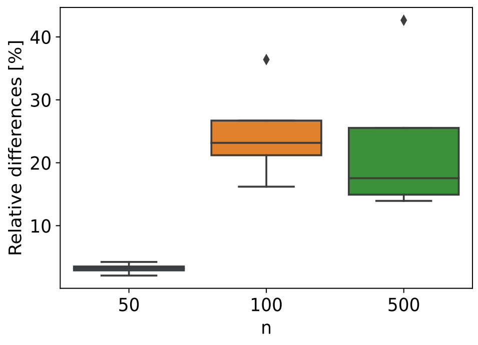
- Concerning the two exact approaches, Coam outperforms Faria significantly for instances with more partitions ($k \geq 10$). Note that the same conclusion is reported in [26]. However, they generally cannot compete with the three heuristic approaches. This is with the exception of Vns in the case of the largest instances, that is, for the instances with $n = 500$ and $k = 20$, where Vns performs poorly.



(a) Instances with $k = 5$.



(b) Instances with $k = 10$.



(c) Instances with $k = 20$.

Figure 1: Plots showing the relative differences (in percent) between the results of Adapt-Cmsa+Ls and the results of the other (four) approaches for instances of benchmark set `A`.

5.3. Statistical evaluation

In order to check the statistical significance of the obtained differences between the studied approaches, we employed the statistical methodology which is outlined below. This concerns the five approaches implemented by us, in addition to Vns from [17] and Ga from [2] (where applicable) and, in the context of instances with $k = 2$, also the specialized approaches iMADEB, MADEB and GRASP as reported in [31]. Initially, Friedman's test was separately executed for all competitor approaches. The results for the considered benchmark sets are divided into four groups with respect to the value of k ($k \in \{2, 5, 10, 20\}$) and separately evaluated. Note that the statistical evaluation is additionally performed in the context of instances with $k \in \{3, 4\}$, where the complete results are only known for a subset of benchmark set `A`; see Table 7.

In those cases in which the null hypothesis H_0 was rejected (H_0 states that there are no statistical differences between the results of the competitor approaches) pairwise comparisons are further performed by using the Nemenyi post-hoc test [22]. The outcome is represented by means of critical difference (CD) plots. In a CD plot, the compared approaches are placed on the horizontal axis according to their average ranking. Thereafter, the CD score is computed for a significance level of 0.05. If the difference is small enough, that is, no statistical difference is detected, a horizontal bar linking statistically equal approaches is drawn.

As already mentioned, the results are divided into five groups: the first group includes the results for all instances from all benchmark sets with $k = 2$; the second group includes the results for instances with $k \in \{3, 4\}$ (only concerning benchmark set `A`); the third group includes the results for the instances with $k = 5$ from all (five) benchmark sets, etc. The respective CD plots are shown in Figures 2-3. The following conclusions may be drawn from there:

- For the instances with $k = 2$, iMADEB from [31] achieves the best average ranking, as shown in Figure 2a. In fact, the ranking of iMADEB is slightly better than the one of MADEB [32]. The third-best approach concerning the average ranking is Ga–Lit. The differences between the results of Ga-Lit and the ones obtained by iMADEB and MADEB are not statistically significant. All other approaches perform statistically worse that the three aforementioned approaches. The results obtained by Ga, Faria, Coam and Adapt-Cmsa+Ls are statistically equivalent. Clearly the worst–performing algorithm on the considered instances is Vns. We want to emphasize at this point that an effective extension of the approaches specialized for $k = 2$ (that is, iMADEB,

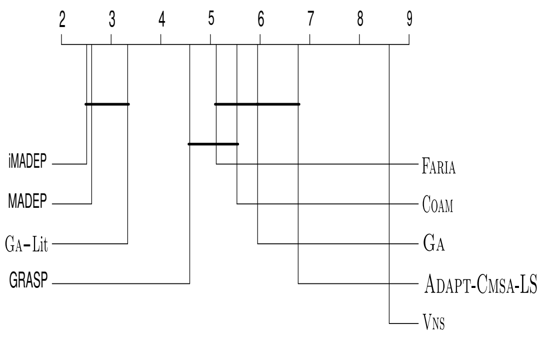
MADEB and GRASP) to the general MDMWNPP ($k \geq 3$) is hardly to achieve as these algorithms are based on a specific binary search space as well as on specific internal procedures involved within the main core of the algorithms. In contrast, despite of weaker results for the instances with $k = 2$, the application of Adapt-Cmsa+Ls is not limited to a specific value of k.

- For the instances with $k \in \{3, 4\}$ on the limited subset of instances from benchmark set A, the best performing algorithm in terms of average ranking as well as solution quality is Adapt-Cmsa+Ls. The second best w.r.t. this criterion is Ga-Lit, which performs statistically equivalent to our re-implementation of Vns. Note that the differences between the best-performing Adapt-Cmsa+Ls and the other approaches are statistically significant. Note also that Vns-Lit and our re-implementation of Vns perform statistically equivalent which gives credibility to our re-implementations; see Figure 2b.

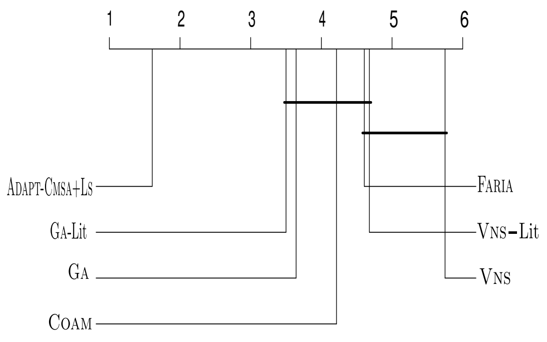
- For the instances with $k = 5$, Adapt-Cmsa+Ls obtains the perfect average ranking score, thus equal to one. The second best is Ga. The differences between these two are statistically significant in favor of Adapt-Cmsa+Ls. Ga and Vns deliver statistically equivalent results. The two approaches based on MILP-solving (Faria and Coam) are the worst-performing ones on this set of instances; see Figure 3a.

- Concerning the instances with $k = 10$, Adapt-Cmsa+Ls again obtains the perfect average ranking score, followed by Ga, see Figure 3b. Similarly as above, the difference between the results obtained by these two approaches are statistically significant in favor of Adapt-Cmsa+Ls. In this case, however, no statistical difference can be detected between the results of Ga and Vns.

- Finally, for the instances with $k = 20$, the same conclusions as above can be drawn concerning Adapt-Cmsa+Ls and Vns. Note that, in this case, Ga outperforms Vns with statistical significance; see Figure 3c.

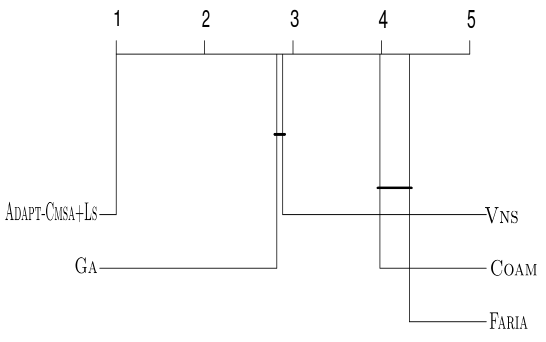


(a) Statistical evaluation on the instances with $k = 2$.

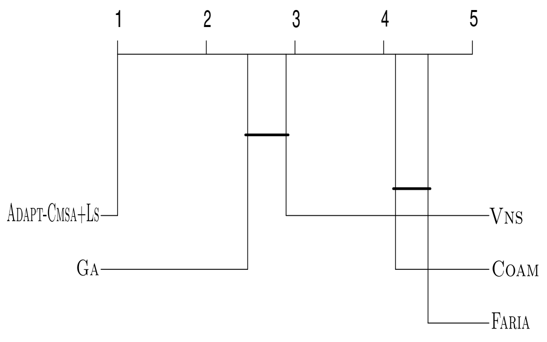


(b) Statistical evaluation on the instances with $k \in \{3, 4\}$.

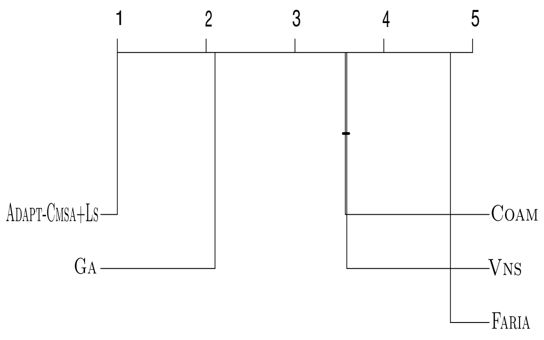
Figure 2: Statistical evaluation of the results for $k \in \{2, 3, 4\}$.



(a) Statistical evaluation for instances with $k = 5$.



(b) Statistical evaluation for instances with $k = 10$.



(c) Statistical evaluation for instances with $k = 20$.

Figure 3: Statistical evaluation concerning the results for $k \in \{5, 10, 20\}$.

790 6. Conclusions and future work

In this paper, we solved the multidimensional multi-way number partitioning problem (MDMWNPP), a generalization of the well-known number partitioning problem. We proposed a MILP–based meta-heuristic algorithm, labelled Adapt-Cmsa+Ls, to solve this problem. The general framework of this algorithm is a more robust version of the effective CMSA meta-heuristic from the literature, additionally equipped with an effective local search procedure. This algorithm is compared to all known competitors from the literature. For the purpose of a rigorous and complete comparison, and due to lack of publicly available source codes, we have carefully re-implemented the Vns and Ga approaches from the literature, which—in the original papers—were only evaluated on a subset of the known benchmark sets. We have provided respective source codes which are made publicly available. Additionally, the results of Vns and Ga from the original papers are also extracted and included in the comparisons under fair conditions whenever applicable. All five known benchmark sets from the literature are used in our experimental evaluation, and the obtained results are additionally statistically evaluated. While Ga, and the two versions of MADEB algorithms from literature significantly outperformed Adapt-Cmsa+Ls in case of instances with a small number of partitions ($k = 2$), the Adapt-Cmsa+Ls algorithm significantly outperformed all approaches from the literature for $k \geq 3$. Thus, this algorithm may be considered the new state-of-the-art algorithm for MDMWNPP in case of the instances with $k \geq 3$.

For future work we plan to further hybridize the Adapt-Cmsa heuristic by replacing the current LS procedure with some other effective, time-efficient LS-based heuristics (such as, iterated LS, or possibly VNS). In that way, solution components of better local optima could be jointly merged, generating more promising subinstances possibly containing new best solutions. Comparing the approaches in the context of real–world benchmark sets seems also interesting as those instances could have some hidden patterns or there might exist internal dependencies between input vectors affecting the main conclusions derived in this paper.

Acknowledgments

C. Blum was supported by grant PID2019-104156GB-I00 funded by MCIN/ AEI/10.13039/501100011033. M. Djukanović is partially supported by Ministry for Scientific and Technological Development, Higher Education and Information Society, Government of Republic of Srpska, B&H under the Project "Development of artificial intelligence methods for solving computer biology problems".

Appendix A. Complete numerical results for the set of instances with $k = 2$

Table A.13: Comparison for instances with $k = 2$ (benchmark set B).

n	m	k	Faria	Coam	Vns	Ga	Adapt-Cmsa+Ls	Ga- Lit	Lit – best
50	2	2	12.465	8.918	20.00	5.50	17.28	9.193	1.42[a]
50	5	2	3212.029	3205.927	4150.72	2351.59	3781.12	3002.869	2609.12[a]
50	10	2	19560.316	21039.450	31401.56	19178.38	24296.65	19560.318	19553.17[b]
50	20	2	56089.632	51917.902	78909.39	53756.40	61815.75	50382.384	55173.79[b]
100	2	2	3.913	14.691	11.97	7.78	21.88	19.513	0.77[a]
100	5	2	2831.944	2086.786	4314.65	2893.53	3517.13	2539.387	2064.91[a]
100	10	2	19374.434	19005.649	31067.38	17919.61	20993.65	18367.389	13589.79[a]
100	20	2	49820.900	46685.387	72276.60	55033.86	59559.65	45673.837	53703.14[b]
500	2	2	4.761	6.281	15.55	256.42	7.91	7.438	0.28[a]
500	5	2	2730.849	3559.877	4719.40	10904.75	2042.73	1453.290	1755.66[a]
500	10	2	20865.981	27029.589	28330.74	34012.78	13577.74	10918.141	12107.12[a]
500	20	2	62739.316	61072.201	69171.82	73564.64	47943.11	38275.503	41464.94[a]
average:			19770.545000	19636.054833	27032.481667	22490.436667	19797.883333	**15850.771833**	16835.2908
#best			0	0	0	2	0	**5**	**5**

Table A.14: Comparison for instances with $k = 2$ (benchmark set C).

n	m	k	Faria	Coam	Vns	Ga	Adapt-Cmsa+Ls	Ga- Lit	Lit – best
50	2	2	12.245	12.245	20.00	5.31	14.79	9.756	1.31[a]
50	5	2	2705.863	2708.911	4150.72	1702.26	1807.43	2655.387	1394.91[a]
50	10	2	17750.997	19619.579	31401.56	18032.39	22095.96	16373.839	17117.30[a]
50	20	2	50560.864	50560.864	78909.39	57424.84	54939.17	51983.338	54141.94[b]
100	2	2	6.443	5.231	11.97	8.82	11.11	6.302	0.62[a]
100	5	2	2854.588	2851.534	4314.65	2166.07	2901.87	2837.741	2751.22[a]
100	10	2	15392.672	15404.882	31067.38	15733.25	17351.70	15262.796	11985.23[a]
100	20	2	47647.322	58128.360	72276.60	54956.14	55220.35	45211.776	51745.85[b]
500	2	2	6.469	8.618	15.55	273.52	12.46	2.539	0.16[a]
500	5	2	2496.388	1903.708	4719.40	10964.42	3010.90	4176.629	1720.18[a]
500	10	2	23184.654	19180.032	28330.74	36030.08	17999.19	10324.870	12801.53[a]
500	20	2	58277.355	63364.387	69171.82	74608.11	54373.45	39474.872	42599.51[a]
average:			18407.988333	19479.029250	27032.481667	22658.767500	19144.865000	**15693.320417**	16354.98
#best			1	1	0	1	0	4	**6**

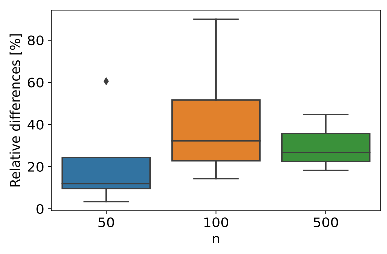
Table A.15: Comparison for instances with $k = 2$ (benchmark set D).

n	m	k	Faria	Coam	Vns	Ga	Adapt-Cmsa+Ls	Ga- Lit	Lit - best
50	2	2	1.703	10.722	13.46	6.39	19.56	5.0230	1.44[a]
50	5	2	2281.898	2275.792	5399.42	2972.43	3297.21	2217.2880	2269.08[a]
50	10	2	14928.077	20636.093	31237.07	19208.93	21261.10	15025.6370	14924.10[b]
50	20	2	53955.965	53955.965	74014.20	59400.54	55862.45	52632.3070	56093.25[b]
100	2	2	5.377	6.433	20.43	5.40	5.61	4.5230	0.00[a]
100	5	2	2982.039	2978.993	5053.36	2767.32	2995.73	2717.6890	2941.45[a]
100	10	2	18291.506	15568.264	30709.94	18513.72	19422.94	17066.3890	14962.60[b]
100	20	2	54986.785	56641.234	74091.93	53309.92	59491.64	46182.3890	50172.85[b]
500	2	2	3.187	16.045	19.62	309.73	14.42	1.0170	0.20[a]
500	5	2	2424.010	1982.865	4659.75	9168.30	2166.34	2083.5850	2.69[a]
500	10	2	23630.800	21291.872	28762.01	33761.08	16719.42	17283.4800	12800.82[a]
500	20	2	63447.344	67054.022	71994.96	74194.41	49428.96	43763.0979	43664.83[a]
average:			236938.6910	242418.3000	325976.1500	273618.1700	230685.3800	16581.8669	**16486.1008**
#best			0	0	0	0	0	4	**8**

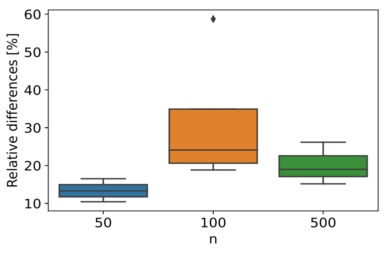
Table A.16: Comparison for instances with $k = 2$ (benchmark set `E`).

n	m	k	FARIA	COAM	VNS	GA	ADAPT-CMSA+LS	GA- Lit	Lit − best
50	2	2	12.696	12.827	19.92	9.34	28.91	7.892	3.23[a]
50	5	2	4682.901	4845.067	4823.40	2584.34	5301.83	4881.732	3959.34[a]
50	10	2	20796.532	19303.636	29506.84	16638.17	20987.76	15263.833	15364.59[b]
50	20	2	48281.499	48281.499	78638.57	59790.25	62510.22	48292.728	51239.31[b]
100	2	2	4.729	1.067	13.99	5.66	6.57	4.241	1.04[a]
100	5	2	3781.746	3778.692	5437.23	2758.43	4245.47	4112.556	2528.46[a]
100	10	2	16516.277	16522.383	27530.33	18907.91	16819.20	15938.556	12180.65[a]
100	20	2	49561.242	63320.091	73060.29	54804.84	53002.13	51822.651	51616.68[b]
500	2	2	7.778	6.080	21.67	396.14	9.70	1.006	0.22[a]
500	5	2	3248.500	4509.678	5425.47	9750.26	2820.35	2023.652	866.96[a]
500	10	2	20221.348	23929.710	28746.59	33721.92	15563.92	15928.492	14854.88[a]
500	20	2	63415.290	57691.895	71039.28	77832.69	57973.37	40654.702	41547.43[b]
average:			19210.878167	20183.552083	27021.965000	23099.995833	19939.119167	16577.670083	**16180.2325**
#best			2	1	0	1	0	3	**8**

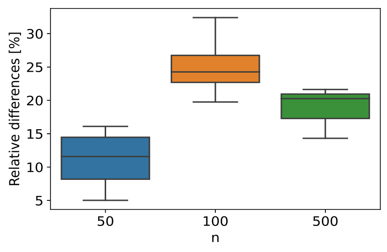
Appendix B. Relative differences between the results of Adapt-Cmsa+Ls and the ones of the other approaches in the context of instances with $k \in \{5, 10, 20\}$



(a) Instances with $k = 5$.

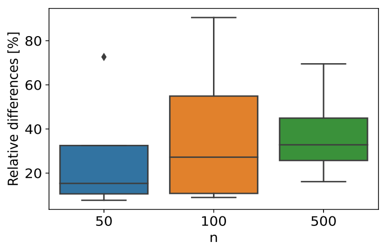


(b) Instances with $k = 10$.

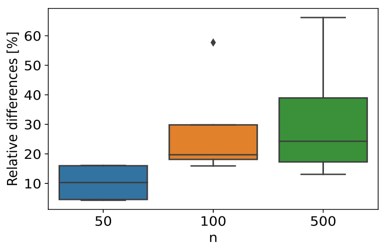


(c) Instances with $k = 20$.

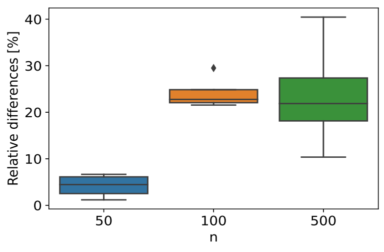
Figure B.4: Plots showing the relative difference (in percent) between the results of ADAPT-CMSA+LS and the best results of the other approaches on benchmark set `B`.



(a) Instances with $k = 5$.

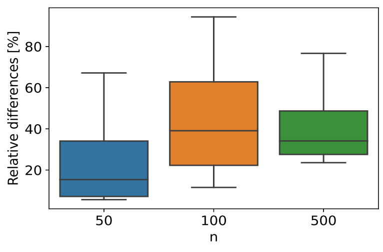


(b) Instances with $k = 10$.

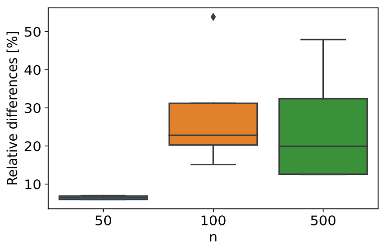


(c) Instances with $k = 20$.

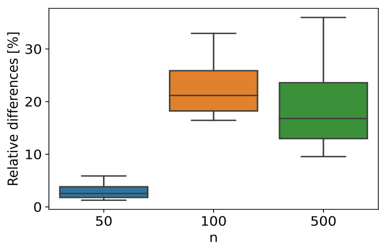
Figure B.5: Plots showing the relative difference (in percent) between the results of Adapt-Cmsa+Ls and the best results of the other approaches on benchmark set C.



(a) Instances with $k = 5$.

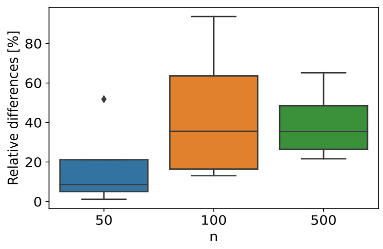


(b) Instances with $k = 10$.

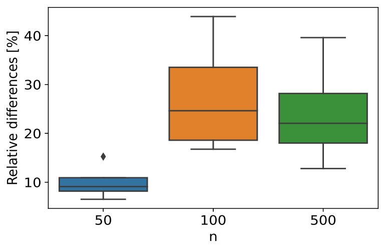


(c) Instances with $k = 20$.

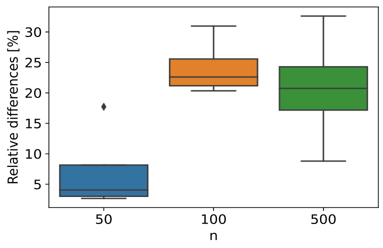
Figure B.6: Plots showing the relative difference (in percent) between the results of Adapt-Cmsa+Ls and the best results of the other approaches on benchmark set D.



(a) Instances with $k = 5$.



(b) Instances with $k = 10$.



(c) Instances with $k = 20$.

Figure B.7: Plots showing the relative difference (in percent) between the results of Adapt-Cmsa+Ls and the best results of the other approaches on benchmark set E.

References

[1] P. C. Pop and O. Matei. A genetic algorithm approach for the multidimensional two-way number partitioning problem. In *International conference on learning and intelligent optimization*, pages 81–86. Springer, 2013.

[2] P. C. Pop and O. Matei. A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem. *Applied Mathematical Modelling*, 37(22):9191–9202, 2013.

[3] H. E. Ben-Smida, F. Chicano, S. Krichen, et al. Construct, merge, solve and adapt for taxi sharing. 2019.

[4] C. Blum. Construct, merge, solve and adapt: application to unbalanced minimum common string partition. In *International Workshop on Hybrid Metaheuristics*, pages 17–31. Springer, 2016.

[5] C. Blum and M. J. Blesa. Construct, merge, solve and adapt: application to the repetition-free longest common subsequence problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 46–57. Springer, 2016.

[6] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* IOP Publishing Ltd., 1997.

[7] C. Blum, P. Pinacho, 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] R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

[9] R. E. Korf. Multi-way number partitioning. In *IJCAI*, volume 9, pages 538–543, 2009.

[10] R. E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106(2):181–203, 1998.

[11] R. E. Korf, E. L. Schreiber, and M. D. Moffitt. Optimal sequential multi-way number partitioning. In *ISAIM*, 2014.

[12] J. Kojić. *Computers & Mathematics with Applications*, 60(8):2302–2308, 2010.

[13] R. Jensi and G. W. Jiji. An enhanced particle swarm optimization with levy flight for global optimization. *Applied Soft Computing*, 43:248–261, 2016.

[14] R. Lewis, D. Thiruvady, and K. Morgan. Finding happiness: an analysis of the maximum happy vertices problem. *Computers & Operations Research*, 103:265–276, 2019.

[15] F. A. Ducha and S. R. de Souza. Algorithms analysis for the number partition problem. *XXXIV CILAMCE*, 2013.

[16] A. F. Faria, S. R. de Souza, and E. M. de Sá. A mixed-integer linear programming model to solve the multidimensional multi-way number partitioning problem. *Computers & Operations Research*, 127:105133, 2021.

[17] A. F. Faria, S. R. de Souza, M. J. F. Souza, C. A. Silva, and V. Nazário Coelho. A variable neighborhood search approach for solving the multidimensional multi-way number partitioning problem. In *International Conference on Variable Neighborhood Search*, pages 243–258. Springer, 2018.

[18] M. A. Akbay, A. López Serrano, and C. Blum. A self-adaptive variant of cmsa: Application to the minimum positive influence dominating set problem. *International Journal of Computational Intelligence Systems*, 15(44):10, 2022.

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

[20] R.Merkle and M.Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

[21] S. Mertens. Number partitioning. *Computational complexity and statistical physics*, page 125, 2006.

[22] T. Pohlert. The pairwise multiple comparison of mean ranks package (pmcmr). *R package*, 27(2020):10, 2014.

[23] J. Kratica, J. Kojić, and A. Savić. Two metaheuristic approaches for solving multidimensional two-way number partitioning problem. *Computers & Operations Research*, 46:59–68, 2014.

[24] C. Bliek1ú, P. Bonami, and A. Lodi. Solving mixed-integer quadratic programming problems with ibm-cplex: a progress report. In *Proceedings of the twenty-sixth RAMP symposium*, pages 16–17, 2014.

[25] B. Alidaee, F. Glover, G. A. Kochenberger, and C. Rego. A new modeling and solution approach for the number partitioning problem. *Journal of Applied Mathematics and Decision Sciences*, 2005(2):113–121, 2005.

[26] B. Nikolic, M. Djukanovic, and D. Matic. New mixed-integer linear programming model for solving the multidimensional multi-way number partitioning problem. *Computational and Applied Mathematics*, 41(3):1–72, 2022.

[27] M. D. Moffitt. Search strategies for optimal multi-way number partitioning. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[28] J. P. Pedroso and M. Kubo. Heuristics and exact methods for number partitioning. *European Journal of Operational Research*, 202(1):73–81, 2010.

[29] E. G. Coffman and G. S. Lueker. *Probabilistic analysis of packing and partitioning algorithms.* Wiley-Interscience, 1991.

[30] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations research*, 39(3):378–406, 1991.

[31] V. Santucci, M. Baioletti, and G. Di Bari. An improved memetic algebraic differential evolution for solving the multidimensional two-way number partitioning problem. *Expert Systems with Applications*, 178:114938, 2021.

[32] V. Santucci, M. Baioletti, G. Di Bari, and A. Milani. A binary algebraic differential evolution for the multidimensional two-way number partitioning problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 17–32. Springer, 2019.

[33] M. F. Argüello, T. A. Feo, and O. Goldschmidt. Randomized methods for the number partitioning problem. *Computers & Operations Research*, 23(2):103–111, 1996.

[34] P. P. Davidson, C. Blum, and J. A. Lozano. The weighted independent domination problem: Integer linear programming models and metaheuristic approaches. *European Journal of Operational Research*, 265(3):860–871, 2018.

[35] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. *Experimental methods for the analysis of optimization algorithms*, pages 311–336, 2010.

[36] N. Karmarkar and R. M. Karp. *The differencing method of set partitioning.* Computer Science Division (EECS), University of California Berkeley, 1982.

[37] F. J. Rodriguez, F. Glover, C. García-Martínez, R. Martí, and M. Lozano. Grasp with exterior path-relinking and restricted local search for the multidimensional two-way number partitioning problem. *Computers & Operations Research*, 78:243–254, 2017.

[38] D. M.Ercegovac and M.Potkonjak. Low-power behavioral synthesis optimization using multiple precision arithmetic. In *Proceedings of the 36th*

Annual ACM/IEEE Design Automation Conference, pages 568–573. Association for Computing Machinery, 1999.

[39] M. Hacibeyoglu, K. Alaykiran, A. M. Acilar, V. Tongur, and E. Ulker. A comparative analysis of metaheuristic approaches for multidimensional two-way number partitioning problem. *Arabian Journal for Science and Engineering*, 43(12):7499–7520, 2018.