A Variable Neighborhood Search Approach for Solving the Multidimensional Multi-Way Number Partitioning Problem

Alexandre Frias Faria[1], Sérgio Ricardo de Souza[1(✉)], Marcone Jamilson Freitas Souza[2], Carlos Alexandre Silva[3], and Vitor Nazário Coelho[4]

[1] Federal Center of Technological Education of Minas Gerais, Belo Horizonte, Brazil
alexandrefrias1@hotmail.com, sergio@dppg.cefetmg.br
[2] Federal University of Ouro Preto, Ouro Preto, Brazil
marcone@iceb.ufop.br
[3] Federal Institute of Minas Gerais, Sabará, Brazil
carlos.silva@ifmg.edu.br
[4] Fluminense Federal University, Niterói, Brazil
vncoelho@gmail.com

Abstract. This paper presents an implementation of the Variable Neighborhood Search (VNS) metaheuristic for solving the optimization version of the Multidimensional Multi-Way Number Partitioning Problem (MDMWNPP). This problem consists in distributing the vectors of a given sequence into k disjoint subsets such that the sums of each subset form a set of vectors with minimum diameter. The proposed VNS for solving MDMWNPP has a good performance over instances with three and four subsets. A comparative study of results found from this proposed VNS and an implementation of Memetic Algorithm (MA) is carried out, running in the same proportional time interval. Although the average results are different, the statistical tests show that results of the proposed VNS are not significantly better than MA in a set of instances analyzed.

Keywords: Multidimensional Multi-Way Number Partitioning Problem · Variable Neighborhood Search · Number Partitioning Problem · Combinatorial optimization

1 Introduction

This paper addresses the Multidimensional Multi-Way Number Partitioning Problem (MDMWNPP), a more general version of the classical Number Partitioning Problem (NPP). This problem is related to any problems involving partitions set like Bin Packing, Machine Scheduling and Clustering, for example.

A. Sifaleras et al. (Eds.): ICVNS 2018, LNCS 11328, pp. 243–258, 2019.
https://doi.org/10.1007/978-3-030-15843-9_19

As it is a generalization, it is necessary to review the problems that originated it to contextualize its study. Throughout this text, a partition of a X set is a collection of mutually disjoint subsets whose union forms X. A k-partition of a set X is a partition of this set, with exactly k non-empty subsets. In this text, the subsets belonging to the partition are called parts. The notation $I_p = \{y \in \mathbb{Z} \ : \ 1 \leq y \leq p\}$ denotes the closed set of all integers between 1 and p.

The Two-Way Number Partitioning Problem (TWNPP) is a well known problem in the literature. Its purpose is to find a 2-partition of the indexes of a V sequence so that the difference between the sums of the elements of each part is minimal. This problem was listed in [7] as one of the basic NP-complete problems, and a series of equivalences between TWNPP and other NP-complete problems are also demonstrated. The first exact algorithms trivially adaptable to TWNPP were presented in [5] and [24], both proposed to solve the Knapsack Problem. In [9], two proposals are presented to transform the heuristics into exact methods of the Branch & Bound type, which are: (i) Complete Greedy Algorithm (CGA), using the Longest Processing Time heuristic (LPT) [4]; and (ii) Complete Karmarkar-Karp Algorithm (CKK), using the Differencing Method, well-known as Karmarkar-Karp Heuristic (KKH) [6]. An exact method based on CKK appears in [14]. In this case, the proposal is to increase the number of prunings in the CKK search tree using a new heuristic called the Balanced Largest Differencing Method (BLDM). Already [19] presents an improvement in the CKK search tree search using a new data structure.

The first generalization of TWNPP is the Multi-Way Number Partitioning Problem (MWNPP), which expands the number of parts in which the sequence indices V must be distributed. Given a numeric sequence V, the goal is to find a k-partition for its indexes, such that the sums of the elements of each part fits into the shortest possible interval. MWNPP is explicitly stated in [6], in which an analysis of Karmarkar-Karp Heuristics (KKH) is presented. This heuristic is focused on the idea of dividing the largest numbers into distinct parts, inserting the differences between the removed elements in the set of unallocated elements as long as this set is not empty. In [2], it is shown that MWNPP is a very difficult problem to be solved by general-purpose metaheuristics, such as Genetic Algorithms, Simulated Annealing, and others. In many cases, these methods have a worse computational cost (in terms of time and performance) when compared to HKK and even to LPT. The exact algorithms presented in [9] were already adapted for the MWNPP.

The first improvement of these works happens with the algorithm Recursive Number Partitioning (RNP), proposed by [10] working with the resolution of minor subproblems derived from MWNPP. Through successive MWNPP conversions of a $(k-1)$-partition to a k-partition, [16] propose an algorithm based on solving smaller subproblems. Currently, the state of the art for the resolution of MWNPP is the Sequential Number Partitioning (SNP) algorithm, presented in [11], and the Cached Iterative Weakening (CIW) algorithm, presented in [23], both fully analyzed in [22]. An application of VNS algorithm for solving MWNPP is described in [1].

The second generalization of TWNPP is the Multidimensional Two-Way Number Partitioning Problem (MDTWNPP). This variant considers a V sequence of vectors of dimension m instead of real numbers, as TWNPP is originally defined. Its purpose is to find a 2-partition of the set of vectors so that the vectors resulting from the sum of each part have minimized the distance induced by the infinite norm. This generalization is initially proposed in [8]. In this same article, a mathematical model in integer linear optimization for MDTWNPP is proposed and solved using CPLEX. This is the only known exact method for solving this problem up to the present moment, according to the knowledge of the authors of this current article. MDTWNPP is also addressed in [20], but in this case using population metaheuristics, such as Memetic Algorithm (MA) and Genetic Algorithm (GA), for its solution. Already [12] presents implementations of Variable Neighborhood Search (VNS) and Electromagnetism-like (EM) metaheuristics to the solution of MDTWNPP and compares the results with those presented in [8] and [20]. The results show that the EM metaheuristic performs slightly superior to the others and strongly superior to the direct solution of the exact model. Another important article addressing MDTWNPP is [21], in which this problem is solved using GRASP+Exterior Path-relinking hybrid metaheuristics. The results obtained, from the same set of instances used in [8,12,20], show the superiority of the proposed procedure.

The third generalization of TWNPP is the Multidimensional Multi-Way Number Partitioning Problem (MDMWNPP). Given a V sequence of vectors of dimension m, the goal of MDMWNPP is to determine a k-partition of vectors such that, added the elements of each part, the diameter of the resulting vectors is minimized. MDMWNPP is originally proposed in [20] with the resolution of three-way ($k = 3$) and four-way ($k = 4$) cases using Memetic Algorithm (MA). It should be stressed that this problem is still little studied in the literature and, on the other hand, is the central object of study of the current article.

This article presents an adaptation of the VNS metaheuristic proposed in [15] for the MDMWNPP solution. The results are compared with those presented in [20] using the same instances of this last article. The justification for applying VNS to MDMWNPP is the set of good results found in [12] for this metaheuristic when solving MDTWNPP.

The article is organized as follows. Section 2 presents the synthetic statement of MDMWNPP and a equivalence proof between the diameter induced by the infinite norm and the objective function introduced in [20]. Section 3 shows the operation of the proposed VNS and the particularity of its neighborhood in rings. Section 4 presents a delineation of the tests performed for the comparison between the results, while Sect. 5 criticizes the results obtained regarding the number of executions required and the form of the instances used for a really valid statistical test. Finally, Sect. 5 concludes the article and presents proposals for future work.

2 Problem Statement

2.1 Fundamental Notions

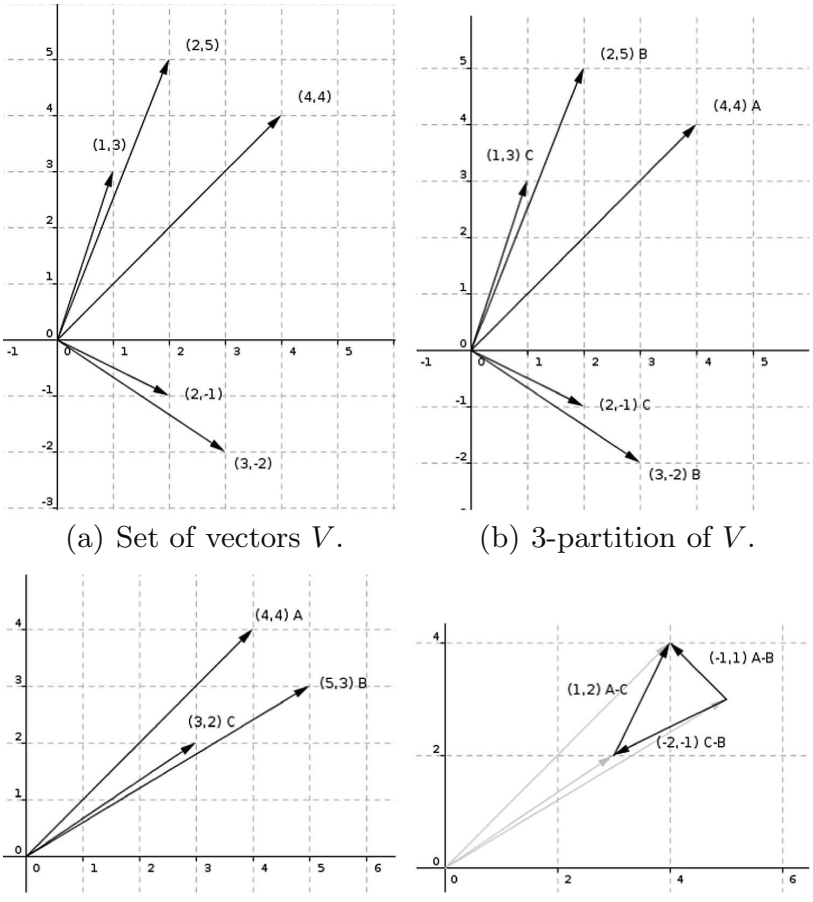
This section introduces fundamental notions concerning MDMWNPP. Let $V = \{v_i\}_{i \in I_n}$ be a set of vectors. The function $gv : P(I_n) \rightarrow \mathbb{R}^m$ receives a discrete subset of vectors in $\mathbb{R}^m$ and returns the sum of its elements. Calculate the function $gv(\cdot)$ as:

$$X \in P(I_n) \quad : \quad gv(X) = \sum_{i \in X} v_i \tag{1}$$

Referring to the l-th coordinate of $gv(X)$, the notation $gv_l(X)$ is used.

Definition 1. *Let $V = \{v_i\}_{i \in I_n}$ be a sequence of vectors such that $v_i \in \mathbb{R}^m$ and k an integer positive number. Find a k-partition of the V indexes, in the form $\{A_j\}_{j \in I_k}$, that minimizes the diameter of the multiset $\{gv(A_j)\}_{j \in I_k}$ given by:*

$$diam_\infty\left(\{A_j\}_{j \in I_k}\right) = \max_{j',j} \{\|gv(A_{j'}) - gv(A_j)\|_\infty\} \tag{2}$$



(a) Set of vectors V.

(b) 3-partition of V.

(c) $\{gv(A), gv(B), gv(C)\}$.

(d) $gv(A) - gv(C)$, $gv(A) - gv(B)$, $gv(C) - gv(B)$.

Fig. 1. Representation of $V = \{(1,3),(4,4),(3,-2),(2,5),(2,-1)\}$ from Example 1.

Example 1. Let $V = \{(1,3),(4,4),(3,-2),(2,5),(2,-1)\}$ be a set of vectors, as shown in Fig. 1(a). This example comes from [20]. Figure 1(b) shows an optimal 3-partition for V. The value of the objective function, from Eq. (2), is $\max_l\{|gv_l(A) - gv_l(C)|, |gv_l(A) - gv_l(B)|, |gv_l(C) - gv_l(B)|\} = 2$.

2.2 Analysis of the Objective Function

The objective function for MDMWNPP proposed in this article, presented in Eq. (2), appears to be different from that originally introduced in [20], given by:

$$f(\{A_j\}_{j\in I_k}) = \max_l \left\{ \left| \max_{j'} gv_l(A_{j'}) - \min_j gv_l(A_j) \right| \right\} \tag{3}$$

In fact, these two functions represent different ways for calculating the diameter of a set of vectors. In the following, an analysis of these two objective functions is presented.

First, it is possible to remove the module from the Expression (3) without any loss, since $\max_{j'} gv_l(A_{j'}) - \min_j gv_l(A_j) \geq 0$ in any case. The idea is to denote the objective function only as the set diameter, allowing a clear interpretation that applies to all variants of the NPP problem (TWNPP, MWNPP, MDTWNPP and MDMWNPP) listed in the current article.

Proposition 1 and Corollary 1 are applied in the demonstration of equivalence of the two expressions in Proposition 2.

Proposition 1. *Let I be a limited real interval. Then:*

$$\max_{x,y\in I} |x - y| = \max_{z\in I} z - \min_{z\in I} z \tag{4}$$

Proof. Since $x, y \in I$, then:

$$\min_{z\in I} z \leq x \leq \max_{z\in I} z \tag{5}$$

$$\min_{z\in I} z \leq y \leq \max_{z\in I} z \tag{6}$$

Manipulating these two expressions, the result is:

$$\min_{z\in I} z - \max_{z\in I} z \leq x - y \leq \max_{z\in I} z - \min_{z\in I} z \tag{7}$$

That is:

$$|x - y| \leq \max_{z\in I} z - \min_{z\in I} z \tag{8}$$

Therefore:

$$\max_{x,y\in I} |x - y| \leq \max_{z\in I}(z) - \min_{z\in I}(z) \tag{9}$$

Equality occurs for the maximum value. This is verified by fixing $x = \max_{z\in I} z$ and $y = \min_{z\in I} z$.

Corollary 1. *Let I be a limited real interval and consider a discrete sequence $\{a_i\}_{i\in I_n} \subset I$. Then:*

$$\max_{i,j\in I_n} |a_i - a_j| = \max_{i\in I_n} a_i - \min_{j\in I_n} a_j \tag{10}$$

Proposition 2. *The distance induced by the infinite norm is given by:*

$$diam_\infty(\{gv(A_j)\}_{j\in I_k}) = \max_{l\in I_m}\left\{\max_{j\in I_k} gv_l(A_j) - \min_{j'\in I_k} gv_l(A_{j'})\right\} \tag{11}$$

Proof. From Definition 1:

$$\begin{aligned}\mathrm{diam}_\infty(\{gv(A_j)\}_{j\in I_k}) &= \max_{j,j'\in I_k} \|gv(A_j) - gv(A_{j'})\|_\infty \\ &= \max_{j,j'\in I_k} \max_{l\in I_m} |gv_l(A_j) - gv_l(A_{j'})| \\ &= \max_{l\in I_m} \max_{j,j'\in I_k} |gv_l(A_j) - gv_l(A_{j'})| \end{aligned} \tag{12}$$

Then, by Corollary 1:

$$\max_{l\in I_m} \max_{j,j'\in I_k} |gv_l(A_j) - gv_l(A_{j'})| = \max_{l\in I_m}\left\{\max_{j\in I_k} gv_l(A_j) - \min_{j'\in I_k} gv_l(A_{j'})\right\} \tag{13}$$

and the proof is finished.

Example 2. Consider the sequence:

$$\begin{aligned}V = \{&(14,48,23),(87,61,48),(76,14,23),(24,25,33),(84,13,49),(25,48,78),\\ &(56,14,73),(55,21,20),(16,13,86),(74,55,31)\}\end{aligned}$$

with $n = 10$ and $m = 3$. For $k \in \{3,4,5\}$, Table 1 shows a feasible (non-optimal) partition and the optimal partition of this sequence V when solving MDTWNPP, with the associated values of objective function. It is worth mentioning that V has a larger dimension than that associated to sequence shown in Example 1. To represent the partitions, the classic coding in [18] is used: a sequence $(s_i)_{i\in I_n}$, where $s_i \in I_k$, indicating the part to which the vector v_i belongs. This notation has some extra details that will be explained in Sect. 3.

Table 1. Example 2: feasible solutions vs optimal solutions

k	Feasible	Obj. val.	Optimal	Obj. val.
3	[1, 1, 1, 2, 3, 3, 2, 3, 1, 2]	54	[1, 2, 1, 2, 3, 3, 2, 1, 1, 3]	22
4	[1, 2, 3, 3, 3, 2, 4, 4, 1, 1]	81	[1, 2, 3, 1, 1, 3, 2, 4, 4, 4]	44
5	[1, 2, 3, 1, 1, 3, 4, 5, 5, 4]	59	[1, 2, 3, 2, 1, 3, 4, 4, 5, 5]	51

In this problem, it should be emphasized not only that the calculation of the objective function is particularly complicated, but also how difficult it is to perform the complete search in the whole space of feasible solutions. According to [25], the number of possibilities for each value of k in Example 2 is given by the Stirling Numbers: $S(10,3)_{k=3} = 9330$, $S(10,4)_{k=4} = 34105$ and $S(10,5)_{k=5} = 42525$, respectively.

3 Proposed Algorithm

The proposed Variable Neighborhood Search (VNS) algorithm works with reallocation moves of an element between parts of the partition $\{A_j\}_{j\in I_k}$. The move $m_{i,j}$ means that an index element $i \notin A_j$ leaves the part where it is and goes to the index part j. This move derives from the enumeration algorithm presented in [18]. It is able to generate all k-partitions of a set.

Consider $s' = \{A'_j\}_{j\in I_k}$ e $s = \{A_j\}_{j\in I_k}$. The neighborhoods $N_1(s)$, $N_2(s)$ e $N_3(s)$ are given by:

$$N_1(s) = \{s' \ : \ s' \leftarrow s \oplus m_{i,j},\ \forall (i,j) \in I_n \times I_k\} \tag{14}$$

$$N_2(s) = \{s' \ : \ s' \leftarrow s \oplus m_{i,j} \oplus m_{i',j'},\ \forall (i,j) \in I_n \times I_k\} \tag{15}$$

$$N_3(s) = \{s' \ : \ s' \leftarrow s \oplus m_{i,j} \oplus m_{i',j'} \oplus m_{i'',j''},\ \forall (i,j) \in I_n \times I_k\} \tag{16}$$

Therefore, these neighborhoods are formed, respectively, by compositions of one, two and three distinct moves $m_{i,j}$. Thus, if $i \in A_l$, a reallocation move $\{A_1, \ldots, A_l, \ldots, A_j, \ldots, A_k\} \oplus m_{i,j}$ leads to $\{A_1, \ldots, A_l - \{i\}, \ldots, A_j \cup \{i\}, \ldots, A_k\}$. The neighborhoods are such that $N_l(s) \cap N_j(s) = \emptyset, \quad \forall i \neq j$.

The encoding used is a vector of size n whose entries are numbers from 1 to k. There are restrictions to these moves, in the form of the following rules:

(i) The index 1 of v_1 must always be in the part 1;
(ii) If v_i is in a part with a single element, the motion $m_{i,j}$ can not be applied;
(iii) A part j will always have at least a i' index less than any index contained in the part $j+1$. This holds for all $j \in I_{k-1}$.

Example 3. Consider the two encodings below:

$$[1,1,1,3,2], \quad [1,1,1,2,3]$$

Note that the first vector does not satisfy the rule (iii) while the second vector satisfies. It is possible to make a move by following rules (i) and (ii) as:

$$[1,1,1,2,3] \oplus m_{2,2} = [1,2,1,2,3]$$

but not:

$$[1,1,1,2,3] \oplus m_{2,3} = [1,3,1,2,3]$$

since $[1,3,1,2,3] = [1,2,1,3,2]$, representing the 3-partition $\{v_1, v_3\}, \{v_2, v_5\}, \{v_4\}$.

Algorithm 1. Codification of solutions s

```
1: V, k, n, m
2: s : s_i                                   ▷ vector (s_i)_{i∈I_n} where s_i ∈ I_k
3: b : b_i          ▷ vector (b_i)_{i∈I_n} where b_1 = 1 and b_i = min{max_{2≤h≤i}{s_h + 1}, k}
4: card : card_j                             ▷ cardinality of the parts
5: sum : sum_j : sum_jl                      ▷ sum_j = gv(A_j) and sum_jl = gv_l(A_j)
```

Algorithm 2. Objective function

```
1: function f(s)
2:     r_1 ← max_j sum_j1 − min_j sum_j1
3:     obj ← r_1
4:     for l ∈ I_m \ {1} do
5:         r_l ← max_j sum_jl − min_j sum_jl
6:         if r_l > obj then
7:             obj ← r_l
8:         end if
9:     end for
10:     return obj
11: end function
```

By rule (iii), the leader element of j can not be passed to a part $j' > j$, because in this case the same solution would have many encodings. The leader element in j can only be moved to $j' < j$ if the second lowest index in j is smaller than all indices of $j + 1$.

These conditions are described in [17], which presents the most efficient known codings to make enumerations of combinatorial structures. There is also a demonstration of the bijection between the set of k-partitions and the coding presented in [18].

With this encoding, the neighborhood size $N_r(s)$ depends on the solution s. In Example 3, there is $N_1([1,1,1,2,3]) = \{[1,2,1,2,3],[1,1,2,2,3]\}$ but, also, $N_1([1,2,1,2,3]) = \{[1,1,1,2,3],[1,2,2,2,3],[1,2,3,2,3],[1,2,1,3,3],[1,2,1,1,3]\}$.

These cardinality differences accumulate as r increases its value to 2 or 3. It is only possible to limit the cardinality of neighborhoods by upper bounds to show that they are polynomials. Thus, consider s being a vector $(s_i)_{i\in I_n}$ and $s_i \in I_k$ representing a partition:

$$|N_r(s)| < \left(\sum_{i \in I_n \setminus \{1\}} \max_{2 \le h \le i} \{s_h\} \right)^r \le \left(\frac{k(2n-k-1)}{2} \right)^r \tag{17}$$

The upper bound shown in Expression (17) is not tight, that is, there is no case where equality occurs, but its expression is compact and already shows that the search space of the neighborhoods used is limited polynomially since $r \le 3$.

The proposed VNS, described in the Algorithm 8, follows the guidelines of [15]. The coding of the solution s is given by Algorithm 1, which holds information essential for the manipulation of the search space and to save computational operations. Instance data can be accessed directly from solution s.

The objective function calculation is done by Algorithm 2. This method is equivalent to the implementation of function (3). The computational cost is

Algorithm 3. Initial solution algorithm

```
1: function LPT(V, k)
2:     for j ∈ I_k do
3:         L_j = 0
4:     end for
5:     l ← rand(I_m)                    ▷ Select one coordinate in I_m for all vectors of V
6:     for i ∈ I_n do
7:         s_i = arg min_j L_j          ▷ build the k-partition
8:         L_{s_i} = L_{s_i} + v_il     ▷ Update sums of the parts
9:     end for
10:    return s
11: end function
```

$\mathcal{O}(\max\{n, km\})$, being $n - k$ operations necessary to obtain the vector *sum* in Algorithm 1 and $(\frac{3}{2}k - 2)(m - 1)$ the number of operations of Algorithm 2.

The initial solution, found by Algorithm 3, is given by an adaptation of the algorithm proposed in [3]. This is a greedy method that fixes a coordinate l and applies a greedy allocation of vectors v_i in the part L_j less loaded at each iteration. A k-partition resulting from this method will be the initial solution of the proposed VNS.

Algorithm 4. Movement of one element

```
1: function move(i, s)
2:     s' ← s
3:     if card_{s_i} = 1 then                ▷ rule 1
4:         return s'
5:     end if
6:     h ← s_i                               ▷ part of element i
7:     for j ∈ I_{b_i} \ {h} do
8:         s_i ← j
9:         sum_j ← sum_j + v_i               ▷ v_i move out from h to j
10:        sum_h ← sum_h − v_i
11:        if f(s) < obj then
12:            obj ← f(s)
13:            s' ← s                        ▷ The new best solution
14:        end if
15:    end for
16:    return s'
17: end function
```

Algorithm 4 returns the best of all possible valid moves of a vector v_i between the possible parts. This procedure is used to enumerate all neighbors in the structures $N_1(s)$, $N_2(s)$ and $N_3(s)$.

Algorithms 5, 6 and 7 show the implementations of the local search method Best Improvement for neighborhoods $N_1(s)$, $N_2(s)$ and $N_3(s)$, respectively. These algorithms explore all the neighbors of a solution s and return the one with the lowest objective function value. The computational complexity of each of them is upperly limited by Expression (17).

Algorithm 5. Best improvement for $N_1(s)$

```
1: function best1(s)
2:     obj ← f(s)                                  ▷ Save objective value of the current solution
3:     for i ∈ I_n do                              ▷ For each v_i the move() is applied
4:         s' ← move(i, s)
5:         if f(s') < obj then
6:             obj ← f(s')
7:             s'' ← s'                            ▷ The new best solution
8:         end if
9:     end for
10:    return s''
11: end function
```

Algorithm 6. Best improvement for $N_2(s)$

```
1: function best2(s)
2:     obj ← f(s)
3:     for i_1 ∈ I_{n-1} do
4:         s' ← move(i_1, s)
5:         for i_2 ∈ I_n \ {I_{i_1+1}} do           ▷ For each tuple (v_{i_1}, v_{i_2}) the move() is applied
6:             s'' ← move(i_2, s')
7:             if f(s'') < obj then
8:                 obj ← f(s'')
9:                 s''' ← s''
10:            end if
11:        end for
12:    end for
13:    return s'''
14: end function
```

Algorithm 7. Best improvement for $N_3(s)$

```
1: function best3(s)
2:     obj ← f(s)
3:     for i_1 ∈ I_{n-2} do
4:         s' ← move(i_1, s)
5:         for i_2 ∈ I_{n-1} \ {I_{i_1+1}} do
6:             s'' ← move(i_2, s')
7:             for i_3 ∈ I_n \ {I_{i_2+1}} do       ▷ For each tuple (v_{i_1}, v_{i_2}, v_{i_3}) move() is applied
8:                 s''' ← move(i_3, s'')
9:                 if f(s''') < obj then
10:                    obj ← f(s''')
11:                    s^(4) ← s'''
12:                end if
13:            end for
14:        end for
15:    end for
16:    return s^(4)
17: end function
```

Algorithm 8 shows the proposed VNS metaheuristic for solving MDTWNPP. The input data are the initial solution, determined by Algorithm 3, and the limit value for runtime. The perturbation in the current solution is a valid random move $m_{i,j}$. The local search uses Best Improvement method to select the neighbor that causes the greatest decrease of objective function and updates it as a current solution, if it is worse than the global solution so far. This local search enumerates the neighbors of a solution using the classical enumeration methods presented in [17] and [18].

Algorithm 8. Adapted VNS algorithm

```
1: function VNS(s, Time)                          ▷ Initial solution and time limit
2:     r ← 1                                      ▷ Initial N_r(s).
3:     s' ← s
4:     while t < Time do                          ▷ Stopping criterion by the time limit
5:         choose s' ∈ N_r(s) at valid random     ▷ Shake
6:         if r = 1 then
7:             s' ← best1(s')
8:         else if r = 2 then
9:             s' ← best2(s')
10:        else
11:            s' ← best3(s')
12:        end if
13:        if f(s') < f(s) then                   ▷ Neighborhood exchange
14:            s ← s'
15:            r = 1
16:        else
17:            r ← 1 + (r mod 3)
18:        end if
19:        count time t
20:    end while
21:    return s, f(s)
22: end function
```

4 Experimental Results

The proposed VNS algorithm was implemented in `C++` language. The computational tests were performed on a computer with Intel Core i7-3770 CPU, 3.4 GHz with 8 cores, 32 GB RAM and Ubuntu 16.04 64-bit operating system using version 3.8 of the `clang` compiler.

Algorithm 8 uses only a single core for its execution. Of the 8 processor cores, only 4 are used simultaneously in sets of distinct instances. The instances used for the experiments are the same used in the articles [8,12,20,21]. The main comparison is with the latest experiments in [20], where MDMWNPP is solved with a Memetic Algorithm.

The goal of the computational experiments of this paper is to compare the result, i.e., the objective function values found by the algorithm, in a same time interval in seconds. However, there is a difference in computational processing capacity between the processor used in the current article and the processor used in [20]. The difference between the single-core performance of the processors used is approximately $\frac{3765}{1109}$, as shown in [13][1,2], which provides benchmarks for processors. In consequence, it is fairer that the experiments in this paper use $\frac{1}{3}$ of the average computational time used in [20] as the time limit. Thus, the values of computacional time shown in Tables 2 and 3 reflect this adjustment factor and are therefore equivalent.

The measures for the comparison of results are based on the average of ten executions. With this average, the relative error measure is given by:

$$Gap(B, A) = \frac{z(A) - z(B)}{z(A)}.100\% \tag{18}$$

[1] https://browser.geekbench.com/processors/748.
[2] https://browser.geekbench.com/processors/309.

This measure shows that the response of the algorithm B is less than that of the algorithm A, when $Gap(B, A) > 0$, where $z(A)$ and $z(B)$ are their respective objective function values on average.

Tables 2 and 3 show the results used in comparing the two methods for $k = 3$ and $k = 4$, respectively. The instances have the form n_ma, following the pattern of Definition 1. Table 4 shows the mean value, standard deviation and p-value of a 95% confidence paired t-test, calculated from the results of the column "Avg. Sol". The hypothesis formulation of the test performed is:

$$\begin{cases} H_1 : \ f_{VNS} < f_{MA} \\ H_0 : \ f_{VNS} \geq f_{MA} \end{cases} \tag{19}$$

For a descriptive analysis of the results, we can observe, in these two tables, the column $Gap(VNS, MA)$. In Table 2 there are eight instances with mark "no", i.e., the MA algorithm was better than Algorithm 8. On the other hand, in Table 3, the number of times the MA beats the Algorithm 8 is six. The difference between the MA and VNS results are, on average, 3359.92 for $k = 3$ and 3509.21 for $k = 4$.

Table 2. Results of [20] vs results from Algorithm 8 for $k = 3$

Instance	MA		VNS		Comparison	
k = 3	Avg. sol	Avg. time	Avg. sol	Avg. time	Gap (VNS, MA)	Better
50_2a	86.2	182.45	130.98	60.94	51.95%	No
50_3a	334.4	202.34	1045.94	67	−212.78%	No
50_4a	3382.5	673.83	2044.01	223.98	39.57%	Yes
50_5a	4125.8	781.82	14266.4	259.97	−245.79%	No
50_10a	37521.6	1189.38	38136.1	395.96	−1.64%	No
50_15a	56015.2	1212.27	68396.1	403.91	−22.10%	No
50_20a	102652	1235.22	92299.1	410.92	10.09%	Yes
100_2a	178.1	342.39	50.7	113.98	71.53%	Yes
100_3a	531.3	428.38	2886.44	141.96	−443.28%	No
100_4a	867.5	673.22	8374.57	223.96	−865.37%	No
100_5a	6224.5	834.62	11527.3	277.97	−85.19%	No
100_10a	47004.8	1436.08	37146.2	477.94	20.97%	Yes
100_15a	96827.3	2073.76	61427.6	690.94	36.56%	Yes
100_20a	113112.5	2564.38	84093.4	853.91	25.66%	Yes

Table 4 reports that there is no significant statistical difference between the averages of the results of the algorithms in the set of tested instances when $k = 3$ and $k = 4$. Even if the average difference between the algorithms is positive, the large standard deviation in the results does not allow to reject the

Table 3. Results of [20] vs results from Algorithm 8 for $k = 4$

Instance	MA		VNS		Comparison	
k = 4	Avg. sol	Avg. time	Avg. sol	Avg. time	Gap (VNS, MA)	Better
50_2a	391.4	342.37	321.69	113.66	17.81%	Yes
50_3a	678.3	536.24	687.75	177.95	−1.39%	Yes
50_4a	836.7	1023.39	3185.66	340.95	−280.74%	No
50_5a	1094.4	1243.28	19979.4	413.87	−1725.60%	No
50_10a	42005.6	1647.76	57025.2	548.83	−35.76%	No
50_15a	56034.7	1843.92	91035.6	613.82	−62.46%	No
50_20a	123627.8	2135.48	108404	710.83	12.31%	Yes
100_2a	687.2	564.02	99.28	187.94	85.55%	Yes
100_3a	1213.7	847.37	5765.02	281.9	−375.00%	No
100_4a	1924.6	922.14	12219.9	306.9	−534.93%	Yes
100_5a	8356.8	1972.39	19056.7	656.78	−128.04%	No
100_10a	75034.8	2819.32	58992.6	938.71	21.38%	Yes
100_15a	122892.7	3193.84	76239.6	1063.82	37.96%	Yes
100_20a	174981.6	4392.01	107619	1463.61	38.50%	Yes

Table 4. Statistical analysis with paired sample for $k \in \{3, 4\}$.

Measures	k = 3	k = 4
Normality	1.63%	6.34%
$p - value_{t-test}$	19.01%	**31.39%**
$p - value_{wilcox-test}$	**47.58%**	54.84%
$\mu_{MA} - \mu_{VNS}$	3359.92	3509.21
$\sigma_{MA,VNS}$	13840.88	26436.14

null hypothesis. The t-test assumes that the data are normally distributed. The Shapiro-Wilk test verifies this condition. The results of the column "Avg. Sol." with $k = 3$ do not satisfy the normality assumption; therefore, the Wilcoxon test was used. For $k = 4$, the t-test can be used.

Table 5 shows the obtained results using the proposed VNS algorithm for solving MDMWNPP to $k \in \{5, 6\}$, i.e., the Multidimensional Five-way and Six-way Multidimensional Number Partitioning Problems, considering the same instances used to solve the cases in which $k \in \{3, 4\}$. The results were obtained considering ten executions for each instance with maximum computational time equal to 1800 s. It is important to highlight that the cases for $k \in \{5, 6\}$ have not been solved previously in any other article, at least according to the knowledge of the authors of the current article.

Table 5. Results for $k \in \{5, 6\}$ with Algorithm 8

VNS	k = 5		k = 6	
Instance	Avg. sol	Avg. time	Avg. sol	Avg. time
50_2a	194.77	1799.31	374.48	1799.7
50_3a	1498.14	1799.96	3073.79	1799.93
50_4a	6678.27	1799.95	9607.94	1799.93
50_5a	11388.1	1799.92	17682.3	1799.93
50_10a	66263.7	1799.49	76327.7	1799.56
50_15a	93023.4	1799.62	93350.7	1799.71
50_20a	113159	1799.58	131340	1799.83
100_2a	618.52	1799.49	393.75	1799.78
100_3a	1229.68	1799.77	2988.6	1799.87
100_4a	11480.8	1799.86	18304.9	1799.77
100_5a	23440.7	1799.45	27570.8	1799.74
100_10a	63401.4	1799.94	76073.5	1799.82
100_15a	98647.3	1799.89	106270	1799.85
100_20a	127963	1799.99	125540	1799.9

5 Conclusion

This article presents a proposal to adapt the VNS metaheuristic to the solution of the Multidimensional Multi-Way Number Partitioning Problem (MDMWNPP). This problem is a generalization of the classical Number Partition Problem (NPP), in which it is assumed that each element of the sequence is a vector and, in addition, k-partitions of the sequence are performed, for $k \geq 2$. Despite this attractive and challenging formulation, this problem remains little studied in the literature. For the purposes of validation of the obtained results, a comparison is made with the only algorithm found in the literature proposed directly to solve the addressed problem, according to the knowledge of the authors of the current article. The proposed VNS algorithm, shown in Algorithm 8, was tested using the same instances used in [20], in which MDMWNPP is solved using Memetic Algorithm. The results are satisfactory as to the quality of the proposed VNS by comparing only the averages and the $gap()$ between the results of the two algorithms, according to Tables 2 and 3. In the statistical analysis, it is not possible to conclude that there is a significant difference between the VNS and the MA in the instances with $k \in \{3, 4\}$.

Specific difficulties were found to support better statistical analysis, such as the low number of executions, and the fact that instances used in this work are dependent on one another. As future work, we intend to apply the VNS metaheuristic combined with mathematical programming formulations for the

solution of MDMWNPP, using, as a test basis, a new group of uniformly distributed generated instances.

Acknowledgements. The authors would like to thank the CAPES Foundation, the Brazilian Council of Technological and Scientific Development (CNPq), the Minas Gerais State Research Foundation (FAPEMIG), the Federal Center of Technological Education of Minas Gerais (CEFET-MG), and the Federal University of Ouro Preto (UFOP) for supporting this research.

References

1. Faria, A.F., de Souza, S.R., Silva, C.A.: Variable neighborhood descent applied to multi-way number partitioning problem. Electron. Notes Discret. Math. **66**, 103–110 (2018). https://doi.org/10.1016/j.endm.2018.03.014. 5th International Conference on Variable Neighborhood Search
2. Gent, I.P., Walsh, T.: Analysis of heuristics for number partitioning. Comput. Intell. **14**(3), 430–451 (1998)
3. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **XLV**(9), 1563–1581 (1966)
4. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**(2), 416–429 (1969)
5. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM (JACM) **21**(2), 277–292 (1974)
6. Karmarkar, N., Karp, R.M.: The differencing method of set partition. Report UCB/CSD 81/113, Computer Science Division, University of California, Berkeley, CA (1982)
7. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Proceedings of a Symposium on the Complexity of Computer Computations. The IBM Research Symposia, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
8. Kojić, J.: Integer linear programming model for multidimensional two-way number partitioning problem. Comput. Math. Appl. **60**(8), 2302–2308 (2010)
9. Korf, R.E.: A complete anytime algorithm for number partitioning. Artif. Intell. **106**(2), 181–203 (1998)
10. Korf, R.E.: Multi-way number partitioning. In: Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI 2009), pp. 538–543 (2009)
11. Korf, R.E., Schreiber, E.L., Moffitt, M.D.: Optimal sequential multi-way number partitioning. In: Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM-2014) (2014)
12. Kratica, J., Kojic, J., Savic, A.: Two metaheuristic approaches for solving multidimensional two-way number partitioning problem. Comput. Oper. Res. **46**, 59–68 (2014)
13. Labs, P.: Geekbench, May 2018. https://browser.geekbench.com/
14. Mertens, S.: A complete anytime algorithm for balanced number partitioning (1999). http://arxiv.org/abs/cs.DS/9903011
15. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
16. Moffitt, M.D.: Search strategies for optimal multi-way number partitioning. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 623–629. AAAI Press (2013)

17. Nijenhuis, A., Wilf, H.S.: Combinatorial Algorithms: For Computers and Calculators. Academic Press, Cambridge (2014)
18. Orlov, M.: Efficient generation of set partitions. Technical report, Faculty of Engineering and Computer Sciences, University of Ulm (2002). http://www.cs.bgu.ac.il/~orlovm/papers/partitions.pdf
19. Pedroso, J.P., Kubo, M.: Heuristics and exact methods for number partitioning. Eur. J. Oper. Res. **202**(1), 73–81 (2010)
20. Pop, P.C., Matei, O.: A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem. Appl. Math. Modell. **37**(22), 9191–9202 (2013)
21. Rodriguez, F.J., Glover, F., García-Martínez, C., Martí, R., Lozano, M.: GRASP with exterior path-relinking and restricted local search for the multidimensional two-way number partitioning problem. Comput. Oper. Res. **78**, 243–254 (2017)
22. Schreiber, E.L.: Optimal Multi-Way Number Partitioning. Ph.D. thesis, University of California Los Angeles (2014)
23. Schreiber, E.L., Korf, R.E.: Cached iterative weakening for optimal multi-way number partitioning. In: Proceedings of the Twenty-Eighth Annual Conference on Artificial Intelligence (AAAI-2014), Quebec City, Canada (2014)
24. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput. **10**(3), 456–464 (1981)
25. Sloane, N.: On-Line Encyclopedia of Integer Sequences (1991). https://oeis.org/