



UNIVERSITÀ
DEGLI STUDI
FIRENZE

FLORE

Repository istituzionale dell'Università degli Studi di Firenze

An improved memetic algebraic differential evolution for solving the multidimensional two-way number partitioning problem

Questa è la Versione finale referata (Post print/Accepted manuscript) della seguente pubblicazione:

Original Citation:

An improved memetic algebraic differential evolution for solving the multidimensional two-way number partitioning problem / Santucci V., Baiocchi M., Di Bari G.. - In: EXPERT SYSTEMS WITH APPLICATIONS. - ISSN 0957-4174. - ELETTRONICO. - 178:(2021), pp. 114938.0-114938.0. [10.1016/j.eswa.2021.114938]

Availability:

The webpage <https://hdl.handle.net/2158/1403611> of the repository was last updated on 2024-12-13T13:23:03Z

Published version:

DOI: 10.1016/j.eswa.2021.114938

Terms of use:

Open Access

La pubblicazione è resa disponibile sotto le norme e i termini della licenza di deposito, secondo quanto stabilito dalla Policy per l'accesso aperto dell'Università degli Studi di Firenze (<https://www.sba.unifi.it/upload/policy-oa-2016-1.pdf>)

Publisher copyright claim:

La data sopra indicata si riferisce all'ultimo aggiornamento della scheda del Repository FloRe - The above-mentioned date refers to the last update of the record in the Institutional Repository FloRe

(Article begins on next page)

An improved Memetic Algebraic Differential Evolution for solving the Multidimensional Two-Way Number Partitioning Problem

Valentino Santucci^a, Marco Baioletti^b, Gabriele Di Bari^b

^a *Department of Humanities and Social Sciences
University for Foreigners of Perugia
Piazza G. Spitella, 3 - Perugia (Italy)
valentino.santucci@unistrapg.it*

^b *Department of Mathematics and Computer Science
University of Perugia
Via Vanvitelli, 1 - Perugia (Italy)
marco.baioletti@unipg.it, gabriele.dibari@unifi.it*

Abstract

In this article, we propose a novel and effective evolutionary algorithm for the challenging combinatorial optimization problem known as Multidimensional Two-Way Number Partitioning Problem (MDTWNPP). Since the MDTWNPP has been proven to be NP-hard, in the recent years, it has been increasingly addressed by means of meta-heuristic approaches. Nevertheless, previous proposals in literature do not make full use of critical problem information that may improve the effectiveness of the search. Here, we bridge this gap by designing an improved Memetic Algebraic Differential Evolution (iMADEB) algorithm that incorporates critical information about the problem. In particular, iMADEB evolves a population of candidate local optimal solutions by adopting three key design concepts: a novel non-redundant bit-string representation which maps population individuals one-to-one to MDTWNPP solutions, a smoother local search operator purposely designed for the MDTWNPP landscapes, and a self-adaptive algebraic differential mutation scheme built on the basis of the Lévy flight concept which automatically regulates the exploration-exploitation trade-off of the search. Computational experiments have been conducted on a widely accepted benchmark suite for the MDTWNPP with a twofold purpose: analyzing the robustness of iMADEB and compare its effectiveness with respect to the state-of-the-art approaches to date for the MDTWNPP. The experimental results provide important indications about iMADEB robustness and, most importantly, clearly show that iMADEB is the new state-of-the-art algorithm for the MDTWNPP.

Keywords: Multidimensional Two-Way Number Partitioning, Algebraic Differential Evolution, Memetic Algorithm, Combinatorial optimization

1. Introduction

The MultiDimensional Two-Way Number Partitioning Problem (from now on abbreviated as MDTWNPP) has been introduced in (Kojić, 2010) as a direct generalization of the classic Number Partitioning Problem (NPP) which has been dubbed “the easiest hard problem” in (Mertens, 2006). In fact, despite the NPP can be stated in very simple terms – given a multiset of positive integers, find a binary partition such that the absolute difference of

the within-set sums is as small as possible (possibly 0) –, it has been proven to be NP-hard in the seminal work of Karp on NP-completeness (Karp, 1972). The MDTWNPP extends the NPP by considering multidimensional vectors of real numbers and the distance induced by the infinity norm rather than, respectively, positive integers and the absolute difference distance.

Formally, an instance of the MDTWNPP is a multiset S of n real-valued vectors of dimension d , i.e., $S = \{v_i \in \mathbb{R}^d : 1 \leq i \leq n\}$, and the goal is to partition S into two subsets S_0 and S_1 such that: $S_0 \cup S_1 = S$, $S_0 \cap S_1 = \emptyset$, and the within-set sums of the vectors in S_0 and S_1 are as close as possible in terms of the L^∞ vector distance that, for two generic vectors $v, w \in \mathbb{R}^d$, is defined as

$$L^\infty(v, w) = \max_{1 \leq j \leq d} |v(j) - w(j)|. \quad (1)$$

Hence, the MDTWNPP objective function to be minimized is

$$f(S_0, S_1) = L^\infty \left(\sum_{v \in S_0} v, \sum_{w \in S_1} w \right). \quad (2)$$

Clearly, the MDTWNPP reduces to the NPP when $d = 1$ and this proves that MDTWNPP is NP-hard as well. Moreover, as noted in (Kojić, 2010), the MDTWNPP misses an important characteristic of the NPP, i.e., the computational complexity of an instance does not decrease together with the ratio between the number of bits required to represent a solution and n – as it has been observed to happen for NPP instances in (Mertens, 2006) and (Corus et al., 2018). Therefore, the MDTWNPP can be considered computationally more difficult than the NPP. This is further confirmed by the experimental analysis conducted in (Rodriguez et al., 2017), where the CPLEX solver, applied to an integer linear programming model for the MDTWNPP, has never been able to improve the trivial lower bound of zero on a set of benchmark instances. Recently, a novel mixed integer linear programming model for the multiway generalization of the MDTWNPP has been proposed and experimented in (Faria et al., 2021). However, its results do not look to be competitive with state-of-the-art results.

For all these reasons, meta-heuristic algorithms have started to be designed for and applied to the MDTWNPP. Notable examples are: (Pop and Matei, 2013b), (Kratika et al., 2014), (Rodriguez et al., 2017), and (Santucci et al., 2019).

Since a MDTWNPP partition $\{S_0, S_1\}$ can be simply represented by an n -length bit-string $x \in \mathbb{B}^n$ in such way that $v_i \in S_{x(i)}$, for $i \in \{1, \dots, n\}$,

the meta-heuristic approaches adopt this simple binary representation for the evolved solutions. However, it is easy to see that such encoding is redundant because, though $\{S_0, S_1\}$ and $\{S_1, S_0\}$ clearly are the same MDTWNPP partition, they are represented by two different bit-strings, one the bitwise negation of the other. Hence, an issue common to all the previously proposed meta-heuristics is that they navigate a search space whose size is double with respect to the number of MDTWNPP solutions.

Furthermore, the most effective MDTWNPP algorithms to date adopt one or more local search operators as part of their main search scheme. Though some of these operators rely on solution neighborhoods purposely designed for the problem at hand – like, for instance, the local search scheme proposed in (Rodriguez et al., 2017) –, none of them fully consider the intrinsic characteristics of the MDTWNPP objective function.

Both these issues – the redundant representation and local search neighborhood design –, if suitably addressed, could allow further advancements in the MDTWNPP literature. In this work, we address these two aspects by introducing: a novel non-redundant bit-string representation which halves the size of the search space navigated by the algorithm, and an efficient local search scheme which allows a smoother local exploration by means of a novel restricted neighborhood built on the basis of the L^∞ distance considered in the MDTWNPP objective function formulation.

These two key ingredients are incorporated in iMADEB: an improved variant of our previously proposed Memetic Algebraic Differential Evolution for the Binary space (Santucci et al., 2019). Like its predecessor, iMADEB adopts a memetic approach which combines a discrete Differential Evolution (DE) global search scheme with a variable neighborhood descent as local search operator. The discrete DE part is built on the basis of a solid algebraic framework for combinatorial optimization (Santucci et al., 2020) which is extended in this work in order to handle the novel reduced bit-string representation. Moreover, we also introduce a Lévy flight-based self-adaptation scheme in order to better regulate the exploration-exploitation trade-off of the search and to improve the ability of the algorithm in escaping stagnation states. Finally, the variable neighborhood descent adopts the newly designed restricted neighborhood and also introduces, with respect to (Santucci et al., 2019), a probabilistic application strategy and a different neighborhood exploration scheme.

A thorough experimental analysis is performed using a widely adopted benchmark suite for the MDTWNPP with a twofold purpose. First, we

analyze the robustness of iMADEB and the impact of its different algorithmic components and, second, we compare iMADEB with the state-of-the-art MDTWNPP meta-heuristics to date.

The rest of the article is organized as follows. Section 2 provides a thorough review of the previous meta-heuristic proposals for the MDTWNPP together with a short description of the original Differential Evolution scheme. Section 3 introduces the novel non-redundant representation and the main scheme of iMADEB. The algebraic differential mutation operator, purposely redesigned for the new bit-string representation, is described in Section 4, the Lévy flight step-size adaptation is depicted in Section 5, while Section 6 describes the variable neighborhood descent procedure. Experimental results are provided and discussed in Section 7, while conclusions are drawn in Section 8 where future lines of research are also depicted.

2. Related work

In Section 2.1 we provide a detailed review of all (to the best of our knowledge) the meta-heuristic proposals for the MDTWNPP to date, while in Section 2.2 we briefly recall the original Differential Evolution scheme together with its main applications.

2.1. Meta-heuristic proposals for the MDTWNPP

An integer linear programming formulation of the MDTWNPP has been originally proposed in (Kojić, 2010), where a set of 210 benchmark instances have been randomly generated and solved by using the linear programming solver CPLEX.

Although the MDTWNPP is a generalization of the NPP, many techniques used to solve the latter cannot be extended to the multidimensional case. For instance, both the NPP greedy algorithm (Mertens, 2006) and the Karmarkar-Karp heuristic (Karmarkar and Karp, 1983) require to sort the set of numbers in input but, in the MDTWNPP, the set of vectors does not admit, in general, a well defined total order. Therefore, meta-heuristic approaches purposely designed for the MDTWNPP have been proposed.

The first of such proposals has been the genetic algorithm (GA) introduced in (Pop and Matei, 2013a) and designed as follows: candidate solutions are represented using the simple redundant bit-string representation described in Section 1; parent individuals for the one-point crossover are selected by means of a binary tournament; mutation works by flipping, with

107 probability 0.1, every bit of an offspring, which is further improved by a
 108 purposely defined heuristic operator; finally, the (μ, λ) replacement strategy
 109 is adopted. The GA outperformed CPLEX in the largest instances with
 110 $n \geq 400$.

111 This GA has been further improved by the same authors in (Pop and
 112 Matei, 2013b), where a memetic algorithm (MA) is proposed for solving a
 113 “multiway” generalization of the MDTWNPP in which the vectors can be
 114 partitioned in $p \geq 2$ subsets. The MA extends the genetic algorithm by
 115 introducing a local search improvement step in such a way that the evolved
 116 population is constantly formed by local optima individuals. The local search
 117 method explores, in succession, three different k -change neighborhoods, for
 118 $k = 1, 2, 3$, where k denotes the number of bits changed by any single move.
 119 However, all the neighborhoods are syntactically defined on the redundant
 120 bit-string representation and, furthermore, they are merely syntactic and do
 121 not consider any intrinsic characteristics of the MDTWNPP. Computational
 122 experiments performed for the case $p = 2$, i.e., in the MDTWNPP prob-
 123 lem, show that MA outperforms both the GA and CPLEX in almost all the
 124 benchmark instances.

125 Two other meta-heuristics have been introduced in (Kratika et al., 2014).

126 The first one is a VNS-like procedure (Mladenović and Hansen, 1997)
 127 which operates on an incumbent solution x , represented as a bit-string (again,
 128 using the redundant encoding). A series of increasing neighborhoods $N_k(x)$
 129 are employed in the shaking phase, along with a local search whose ele-
 130 mentary step is to flip both a 0-bit and a 1-bit of x . This corresponds to
 131 simultaneously swap two vectors: one vector moves from the set S_0 to the set
 132 S_1 , while the other one moves in the opposite direction. The generic neigh-
 133 borhood $N_k(x)$ is defined as the set of all the bit-strings having Hamming
 134 distance k from x thus, as before, the neighborhoods are merely syntactic.
 135 The parameter k is increased, from 2 to $\min\{30, \lfloor n/4 \rfloor\}$, circularly at every
 136 iteration where the produced local optimum does not improve the incumbent
 137 solution.

138 The second meta-heuristic uses an Electromagnetism-like (EM) approach.
 139 A solution is represented as a real vector in $[0, 1]^n$, which is decoded to a bi-
 140 nary partition by means of a simple thresholding procedure: the vector v_i
 141 of the MDTWNPP instance is assigned to the set S_0 when the i -th solution
 142 component is smaller than 0.5, otherwise it is assigned to S_1 . At each gener-
 143 ation, every individual undergoes to local search and scaling operators, then
 144 all the solutions are moved according to “electromagnetic forces” that can

145 be attractive or repulsive depending on the objective values in the current
 146 population. It is worthwhile to note that the real vector encoding of EM is
 147 highly redundant because any MDTWNPP solution may be represented by
 148 an infinite number of real vectors.

149 The experiments conducted in (Kratika et al., 2014) show that VNS
 150 and EM obtained comparable performances and both outperform MA and
 151 CPLEX.

152 A GRASP procedure for the MDTWNPP, equipped with an Exterior
 153 Path Relinking method, is described in (Rodriguez et al., 2017). The algo-
 154 rithm evolves a set of solutions, called “elite set”. At each step, the GRASP
 155 procedure produces a new solution by means of two operations: construction
 156 and local improvement. The former operation builds-up a solution by means
 157 of a greedy method, while the latter iteratively improves the incumbent solu-
 158 tion by using a (possibly restricted) local search in the space of the 2-change
 159 neighborhood. Then, the Path Relinking phase explores a path from the new
 160 solution s_i to a randomly selected solution s_G in the elite set (Interior PR) or
 161 beyond s_G (Exterior PR), returning the best solution found in the path. The
 162 configuration with the Exterior Path Relinking, i.e., GRASP+ePR, reached
 163 better performances and outperformed both VNS and CPLEX. Importantly,
 164 the restricted neighborhood of GRASP+ePR is the first proposal which tries
 165 to consider the intrinsic characteristics of the MDTWNPP. In fact, the neigh-
 166 borhood is restricted by considering the closer pairs of vectors from different
 167 subsets but, unfortunately, the authors used the Euclidean distance and not
 168 the L^∞ distance considered in the definition of the MDTWNPP objective
 169 function.

170 Other two works related to the MDTWNPP have been proposed in (Hacibe-
 171 yoglu et al., 2014) and (Hacibeyoglu et al., 2018). The former proposes a
 172 greedy heuristic and a genetic algorithm, but only for the special case of bi-
 173 dimensional vectors (i.e., $d = 2$), while the latter describes an experimental
 174 comparison of four meta-heuristic methods: another genetic algorithm, sim-
 175 ulated annealing, migrating bird optimization algorithm and clonal selection
 176 algorithm. However, neither (Hacibeyoglu et al., 2014) nor (Hacibeyoglu
 177 et al., 2018) present experimental results which are competitive with the
 178 previously described proposals.

179 To the best of our knowledge, the state-of-the-art MDTWNPP algorithm
 180 to date is MADEB, i.e., the memetic algebraic differential evolution proposed
 181 in our preliminary work (Santucci et al., 2019) of which this article is an
 182 extension. In fact, MADEB significantly outperformed GRASP+ePR, VNS

183 and CPLEX on a set of 126 benchmark instances by obtaining the best
 184 average results on 106 instances (about 84% of the benchmark suite) and
 185 76 new best known solutions. However, also MADEB adopts the simple
 186 redundant bit-string representation and the merely syntactic neighborhood
 187 definitions for its local search part.

188 Summarizing, all the meta-heuristics proposed in the MDTWNPP liter-
 189 ature adopt a redundant solution representation and the most effective ones
 190 use local search operators that, however, do not take into account the intrinsic
 191 characteristics of the MDTWNPP objective function. We believe that
 192 addressing these two aspects may bring to more effective methods. Hence, in
 193 this work, we extend MADEB by incorporating a novel non-redundant repre-
 194 sentation, a smoother local search procedure, and a self-adaptive mechanism
 195 to control the exploration-exploitation balance of the search. Moreover, we
 196 provide a more thorough experimental analysis on all the 210 benchmark
 197 instances originally proposed in (Kojić, 2010).

198 2.2. Differential Evolution

199 Differential Evolution (DE) is a population based evolutionary meta-
 200 heuristic, originally proposed in (Storn and Price, 1997), for continuous op-
 201 timization problems.

202 DE evolves a population of N real vectors $\{x_1, \dots, x_N\}$ by iteratively ap-
 203 plying three genetic operators: differential mutation, crossover and selection.

The key operator of DE is the differential mutation which, for every
 population individual x_i , produces a mutant vector y_i as a linear combination
 of few other population individuals. Formally,

$$y_i \leftarrow x_{\text{base}} + F \cdot (x_{r_1} - x_{r_2}), \quad (3)$$

204 where: x_{r_1} and x_{r_2} are two randomly selected population individuals different
 205 between them and with respect to x_{base} which, depending on the chosen
 206 mutation strategy, may be set to: the current individual x_i , another random
 207 population individual, or the best solution so far. Moreover, $F > 0$ is the DE
 208 scale factor parameter which is usually tuned offline or online by means of self-
 209 adaptive mechanisms such as (Brest et al., 2006) or (Tanabe and Fukunaga,
 210 2013).

211 Notably, equation (3) perturbs x_{base} by an amount that is obtained from
 212 the differences' distribution of the DE population which is, itself, constantly
 213 evolved during the search. This mechanism allows DE to continuously adapt

214 its mutation strength and it is the reason of why the differential mutation is
 215 usually considered the core operator of DE (Price et al., 2006).

After the differential mutation, any population individual x_i undergoes a crossover phase with its corresponding mutant y_i . Though many different crossover strategies have been proposed (Storn and Price, 1997; Price et al., 2006), the most used one is the binomial crossover scheme which, for every dimension j , produces an offspring z_i according to

$$z_i(j) \leftarrow \begin{cases} y_i(j) & \text{if } r_j < CR \text{ or } j = t, \\ x_i(j) & \text{otherwise,} \end{cases} \quad (4)$$

216 where: r_j is a randomly generated number in $[0, 1)$, t is a dimension randomly
 217 selected for each individual and ensuring that at least one component of the
 218 mutant is inherited by z_i , while $CR \in [0, 1]$ is the DE crossover probability
 219 which is often self-adapted as, for instance, in (Brest et al., 2006) or (Tanabe
 220 and Fukunaga, 2013).

221 Once z_i is generated, it competes with x_i in order to enter the next
 222 generation population. In the most used selection scheme, the fitter between
 223 z_i and x_i is selected.

224 During the years, DE has been applied to a variety of problems and fields
 225 such as, among the others, product line design (Tsafarakis et al., 2020), com-
 226 putational systems biology (Penas et al., 2015), time series forecasting (Wang
 227 et al., 2015), image segmentation (Cuevas et al., 2010), underwater glider
 228 path planning (Zamuda and Sosa, 2019), traffic signal control (Bi et al.,
 229 2014), and memetic computing (Piotrowski, 2013). Moreover, interesting
 230 variants of DE for combinatorial optimization problems have been proposed
 231 in (Santucci et al., 2016; Baioletti et al., 2020, 2018).

232 3. Main scheme of iMADEB

233 iMADEB is a memetic algebraic differential evolution which improves our
 234 previous proposal (Santucci et al., 2019) by extending it along three different
 235 lines: non-redundant bit-string representation, Lévy flight mutation, and
 236 redesigned local search procedure.

237 The focal point for the non-redundant representation is that a generic
 238 partition $\{S_0, S_1\}$ is equivalent to $\{S_1, S_0\}$. In fact, under the objective
 239 function definition provided in equation (2), $f(S_0, S_1) = f(S_1, S_0)$. Though
 240 this aspect is easy to read, all the previous population-based meta-heuristics
 241 for the MDTWNPP (described in Section 2) do not seem to address this

point. In iMADEB we force the genotypic representation to be in one-to-one relationship with the problem phenotype – binary partitions without an ordering of the subsets – by encoding any solution with a string of $n - 1$ bits and adopting the convention that the n -th vector of the MDTWNPP instance resides in the first lexicographic subset of the partition.

Formally, given the set $S = \{v_1, v_2, \dots, v_n\}$ of the n instance vectors, the “shortened” bit-string $x \in \mathbb{B}^{n-1}$ uniquely represents the binary partition (S_0, S_1) of S where

$$\begin{aligned} S_0 &= \{v_i \in S : x(i) = 0\} \cup \{v_n\}, \\ S_1 &= \{v_i \in S : x(i) = 1\}. \end{aligned} \tag{5}$$

Hence, the $(n - 1)$ -length bit-string does not act on v_n which is used as reference vector, while the i -th bit of x decides if v_i belongs to the same set of v_n (when $x(i) = 0$) or not (when $x(i) = 1$). Clearly, any other choice for the reference vector is equivalent.

It is also easy to note that: (i) every bit-string represents now a different partition, and (ii) the size of the genotypic search space is reduced from 2^n to 2^{n-1} solutions.

By using this representation, iMADEB evolves a population of N bit-strings by iterative applications of the following search operators: binary algebraic differential mutation, variable neighborhood descent, and selection. Its main scheme is depicted in Algorithm 1.

The population is initialized following the sparse bit-string initialization proposed in (Santucci et al., 2019), i.e., for every population individual x_i : a random value $p_i \in [0, 1]$ is generated and, for $1 \leq j < n$, $x_i(j)$ is set to 1 with probability p_i , or 0 otherwise. The rationale of this initialization scheme is to generate a more sparse population. In fact, the expected number of 1-bits throughout the population individuals is uniformly distributed in $[0, n - 1]$ (and not fixed to $(n - 1)/2$ as in the classic random initialization).

For every population individual x_i , **AlgebraicDifferentialMutation** generates a mutant y_i as follows:

$$y_i \leftarrow x_i \oplus F \odot (x_{r_1} \ominus x_{r_2}), \tag{6}$$

where: $F > 0$ is a scale factor parameter, x_{r_1} and x_{r_2} are two randomly chosen population individuals different between them and with respect to x_i , and the $\oplus, \ominus, \odot$ are the binary algebraic operators defined and discussed

Algorithm 1 Main scheme of iMADEB

```
1: Initialize  $N$  bit-strings  $x_1, \dots, x_N \in \mathbb{B}^{n-1}$ 
2: while termination condition is not satisfied do
3:   for  $i = 1$  to  $N$  do
4:      $y_i \leftarrow \text{AlgebraicDifferentialMutation}(x_i)$ 
5:      $z_i \leftarrow \text{VariableNeighborhoodDescent}(y_i)$ 
6:   end for
7:   for  $i \leftarrow 1$  to  $N$  do
8:      $x_i \leftarrow \text{Selection}(x_i, z_i)$ 
9:   end for
10:  if  $x_{best}$  was not updated in the last 1000 generations then
11:    Reinitialize the bit-strings in  $\{x_1, \dots, x_N\} \setminus \{x_{best}\}$ 
12:  end if
13: end while
14: return  $x_{best}$ 
```

268 in Section 4 by taking into account the newly introduced non-redundant
269 representation.

270 It is worthwhile to note that the scale factor F regulates the magnitude of
271 the mutation and it is self-adapted during the evolution by means of a newly
272 designed adaptation scheme based on the Lévy flight concept (Viswanathan
273 et al., 1999). The aim is to allow the search to occasionally perform large
274 “jumps” in order to escape from stagnation states. The Lévy flight adapta-
275 tion is described in Section 5.

276 After the differential mutation, the mutant y_i undergoes a local search
277 phase as in other memetic approaches (Moscato et al., 2004; Moscato and
278 Cotta, 2003, 2019). The local search procedure `VariableNeighborhoodDescent`
279 adopts two different neighborhood and generates the trial individual z_i in
280 such a way that z_i is a local optimum of both neighborhoods. With respect to
281 the previous proposal (Santucci et al., 2019), `VariableNeighborhoodDescent`
282 is modified by considering: a new and smoother neighborhood definition, the
283 best-improvement exploration scheme, and a probabilistic application strat-
284 egy. All these aspects are described in Section 6.

285 The `Selection` procedure replaces the population individual x_i with the
286 trial bit-string z_i if and only if $f(z_i) < f(x_i)$, where f is the objective function
287 defined in equation (2). Moreover, in order to escape persistent stagnation
288 states, if the best population individual x_{best} was not updated during the last

289 1000 generations, then all the population, except x_{best} , is reinitialized.

290 4. Binary Algebraic Differential Mutation

291 In iMADEB, the binary algebraic differential mutation is in charge of
 292 exploring the search space by providing new seed solutions to the following
 293 local search phase. As depicted in equation (6), every individual x_i is mutated
 294 by exploiting the discrete difference between other two randomly selected
 295 population individuals (x_{r_1} and x_{r_2}). As in the classic Differential Evolution
 296 (DE) (Storn and Price, 1997), the differences’ distribution evolves together
 297 with the population, thus constantly adapting the exploration strength of
 298 the algorithm during the search.

299 However, classic DE addresses numerical optimization problems and re-
 300 quires a careful redefinition in order to be applied to the binary space.
 301 In (Santucci et al., 2016, 2020), an original algebraic framework has been
 302 introduced in order to design a differential mutation for combinatorial search
 303 spaces in such a way that it consistently simulates the behavior of its nu-
 304 merical counterpart. The framework abstractly defines the design of the
 305 combinatorial differential mutation for any discrete space representable by a
 306 finitely generated group (Santucci et al., 2016, 2020; Baiocchi et al., 2020).

307 In the following, after briefly recalling the algebraic framework for the
 308 binary space as used in the previous MADEB proposal (Section 4.1), we in-
 309 troduce its revisited implementation for the newly introduced non-redundant
 310 binary representation (Section 4.2) and we analyze the search behavior of the
 311 binary differential mutation used in iMADEB (Section 4.3).

312 4.1. Previous algebraic operators for the binary space

313 In order to define the operators $\oplus, \ominus, \odot$ for the bit-strings, the abstract
 314 algebraic framework described in (Santucci et al., 2020) requires: (i) a binary
 315 operation in $\mathbb{B}^n$ which satisfies the group properties, (ii) a subset of generator
 316 bit-strings which generates all the other bit-strings, and (iii) a fast factoriza-
 317 tion algorithm which decomposes any bit-string in terms of generators.

318 By denoting the bitwise XOR operation with $\underline{\vee}$, it is easy to see that $\mathbb{B}^n$
 319 forms a group under $\underline{\vee}$. In fact, $\underline{\vee}$ is commutative and associative, the “all
 320 zeros” bit-string $\mathbf{0}$ is the neutral element, and the inverse of any $x \in \mathbb{B}^n$ is
 321 itself, i.e., $x^{-1} = x$.

322 Given $x, y \in \mathbb{B}^n$, we recall that: the Hamming weight $|x|$ is the number
 323 of 1-bits in x , and the Hamming distance between x and y is $|x \vee y|$, i.e., the
 324 number of positions i such that $x(i) \neq y(i)$.

325 $\mathbb{B}^n$ is finitely generated by the generating set $U \subset \mathbb{B}^n$ composed by the n
 326 bit-strings with Hamming weight equal to 1, i.e., any generator $u_i \in U$, for
 327 $1 \leq i \leq n$, is such that $u_i(i) = 1$, while the rest of its bits are 0. Therefore,
 328 any $x \in \mathbb{B}^n$ can be written as $x = u_{i_1} \vee u_{i_2} \vee \dots \vee u_{i_l}$, where $i_1, i_2, \dots, i_l$
 329 are the indexes of the 1-bits of x . Clearly, $l = |x|$. The decomposition is
 330 minimal and unique, up to reordering the indexes $i_1, i_2, \dots, i_l$. We exploit this
 331 property and we represent the minimal decomposition of $x \in \mathbb{B}^n$ as the set
 332 $U_x = \{u_i \in U : x(i) = 1\}$. Note anyway that any ordering of the generators
 333 in U_x is a sequence that fulfills the abstract framework definitions (Santucci
 334 et al., 2020). Importantly, for each $x \in \mathbb{B}^n$, the application of the generator
 335 u_i to x , i.e., $x \vee u_i$, corresponds to flipping the i -th bit of x .

336 Therefore, by following the abstract definitions given in (Santucci et al.,
 337 2020) and (Santucci et al., 2016), it is now possible to concretely derive the
 338 operations $\oplus, \ominus, \odot$ for the binary space.

339 Given $x, y \in \mathbb{B}^n$, the addition $\oplus$ is defined as $x \oplus y := x \vee y$, while the
 340 subtraction uses the property that $x^{-1} = x$ and, therefore, it coincides with
 341 the addition, i.e., $y \ominus x := x \vee y$.

342 Given a scalar $F \geq 0$ and a bit-string $x \in \mathbb{B}^n$, the stochastic scalar
 343 multiplication $z = F \odot x$ is defined as randomly selecting a $z \in \mathbb{B}^n$ such
 344 that its decomposition U_z : (i) has size $k = \lceil F \cdot |x| \rceil$, and (ii) when $F \leq 1$,
 345 $U_z \subseteq U_x$, while (iii) if $F > 1$, $U_z \supseteq U_x$. It is easy to see that any ordering
 346 of the generators in U_z satisfies the abstract scalar multiplication properties
 347 depicted in (Santucci et al., 2020). Operatively, when $F \leq 1$, U_z is randomly
 348 selected among the $\binom{|x|}{k}$ subsets of size k of U_x while, when $F > 1$, U_z is
 349 computed as $U_x \cup A$, where A is randomly selected among the $\binom{n-|x|}{k-|x|}$ subsets
 350 of size $k - |x|$ of $U \setminus U_x$. Note also that $|F \odot x|$ cannot be larger than n , thus
 351 we limit F to $\frac{n}{|x|}$ when larger.

352 As any other finitely generated group, $(\mathbb{B}^n, \vee, U)$ has an associated Cayley
 353 graph that, in our case, is the binary hypercube with n vertices, where all
 354 the pairs of bit-strings, differing in a single bit i , are connected by an edge
 355 labelled with $u_i \in U$. Hence, it is easy to see that the Cayley graph is the
 356 usual binary search space whose neighborhood is induced by bit-flip moves.
 357 Moreover, it is also possible to show that the operations $\oplus, \ominus, \odot$ simulate – in
 358 the binary space – the behavior of their numerical counterparts on the classic

Euclidean space. The main idea is that the dichotomic interpretation of a Euclidean vector, both as point and as displacement (between two points), is brought to the binary Cayley graph by considering a bit-string both as a vertex and as a shortest path (between two vertices). For further details about the algebraic framework we refer the interested reader to (Santucci et al., 2020).

4.2. Algebraic operators for the non-redundant binary space

Given $x \in \mathbb{B}^n$, we denote by x' its bitwise negation. For example, let $x = (1001)$, then $x' = (0110)$. As discussed in Section 3, using the trivial bit-string representation for the MDTWNPP, we have that both x and x' correspond to exactly the same binary partition, thus they represent the same phenotypic solution. For this reason, the new non-redundant representation which fixes a reference vector and works with $m = n - 1$ bits is introduced. In this way, x and x' represent two different MDTWNPP partitions and the mapping between genotype and phenotype is now one-to-one.

However, directly applying the previous algebraic operators to the reduced representation results in a subtle issue. Let see it with a small example: consider $n = 4$ (thus $m = 3$ and the reference vector is v_4) and the m -length bit-string $x = (000)$ together with its negation $x' = (111)$. Under the non-redundant representation, x and x' respectively represent the following two partitions:

$$\begin{aligned} (S_0^x &= \{v_1, v_2, v_3, v_4\}, & S_1^x &= \emptyset), \\ (S_0^{x'} &= \{v_4\}, & S_1^{x'} &= \{v_1, v_2, v_3\}). \end{aligned}$$

Clearly, x and x' are distant three bit-flips, but (S_0^x, S_1^x) can be transformed to $(S_0^{x'}, S_1^{x'})$ by simply changing the subset of the reference vector v_4 and remembering the naming convention that the first lexicographic subset of a partition is the one which includes the reference vector. More in general, we have that the phenotypic distance between two solutions can be much smaller than the Hamming distance between the corresponding bit-strings.

Fortunately, our algebraic framework allows to address this issue in an elegant way. The only modification is to add the “all ones” bit-string $\mathbf{1}$ to the generating set. Formally, we consider the generating set $\hat{U} \subset \mathbb{B}^m$ which is defined as $\hat{U} = U \cup \{\mathbf{1}\}$. Therefore, the generators in $\hat{U}$ are: the “all ones” bit-string $\mathbf{1}$ and the m bit-strings with a single 1-bit (i.e., those in U).

This simple modification introduces shortcuts in the Cayley graph in such a way that the genotypic distance between two solutions exactly corresponds

387 to their phenotypic distance. By considering the previous example, we have
 388 that $x' = x \oplus \mathbf{1} = x \vee \mathbf{1}$, i.e., x' can be obtained from x by a single geno-
 389 typic move, exactly as it happens between their corresponding partitions.
 390 Summarizing, the $\mathbf{1}$ generator semantically means “change the subset of the
 391 reference vector v_n ”, while the other $n - 1$ generators u_i , with $1 \leq i < n$,
 392 mean “change the subset of vector v_i ”.

393 We graphically compare the Cayley graphs of the redundant and non-
 394 redundant representations by showing them in, respectively, Figures 1 and 2.

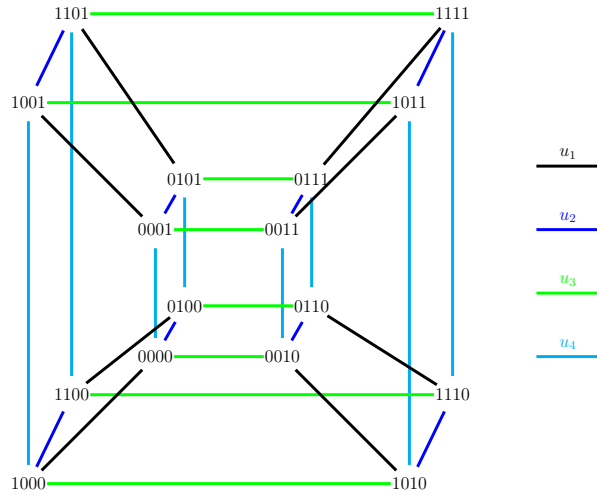


Figure 1: Cayley graph G_1 for the redundant repr. (i.e., using the generating set U)

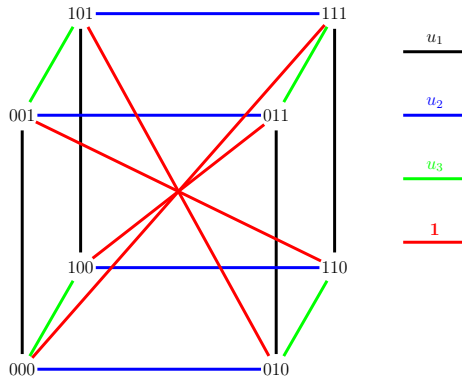


Figure 2: Cayley graph G_2 for the non-redundant repr. (i.e., using the generating set $\hat{U}$)

Figure 1 depicts the Cayley graph G_1 of the redundant representation for the case $n = 4$. The edge colors correspond to the generators in U : black is u_1 , blue is u_2 , green is u_3 , and cyan is u_4 . For instance, the solution (0101) is connected with the blue edge to the solution (0101) $\vee u_2 = (0001)$.

Figure 2 depicts the Cayley graph G_2 of the non-redundant representation for the same search space with four MDTWNPP vectors. Here, all the bit-strings have length three. The edge colors correspond to the generators of the new generating set $\hat{U}$: black is u_1 , blue is u_2 , green is u_3 , and the newly introduced generator **1** is depicted in red. Importantly, each vertex in G_2 corresponds to two vertices in G_1 . For instance, the vertex (000) in G_2 corresponds to the vertices (0000) and (1111) in G_1 , because both represents the partition $\{\{v_1, v_2, v_3, v_4\}, \emptyset\}$. Moreover, confirming our previous example, we have that, in G_2 , the vertex (111) is now only one edge away from (000).

Importantly, since the generating set is slightly changed, a new factorization algorithm for $(\mathbb{B}^m, \vee, \hat{U})$ is required. Anyway, it is a simple modification of what is described in Section 4.1. Its working scheme is provided in Algorithm 2 and described as follows. Given the $x \in \mathbb{B}^m$ in input, lines 2–3

Algorithm 2 Factorization algorithm for the non-redundant representation

```

1: function FACTORIZATION( $x \in \mathbb{B}^m$ )
2:    $t_0 \leftarrow$  number of 0-bits in  $x$ 
3:    $t_1 \leftarrow$  number of 1-bits in  $x$ 
4:   if  $t_1 \leq t_0$  then
5:      $\hat{U}_x \leftarrow \{u_i \in U : x(i) = 1\}$ 
6:   else if  $t_1 > t_0 + 1$  then
7:      $\hat{U}_x \leftarrow \{\mathbf{1}\} \cup \{u_i \in U : x(i) = 0\}$ 
8:   else ▷ Here  $t_1 = t_0 + 1$ 
9:      $r \leftarrow$  random value in  $[0, 1)$ 
10:    if  $r < 0.5$  then
11:       $\hat{U}_x \leftarrow \{u_i \in U : x(i) = 1\}$ 
12:    else
13:       $\hat{U}_x \leftarrow \{\mathbf{1}\} \cup \{u_i \in U : x(i) = 0\}$ 
14:    end if
15:  end if
16:  return  $\hat{U}_x$ 
17: end function

```

calculate in t_0 and t_1 the number of, respectively, 0-bits and 1-bits of x . Then, the factorization $\hat{U}_x$ is computed as the shorter between: (i) taking

414 the generators from U which correspond to the positions of the 1-bits in x
 415 (lines 4–5), and (ii) considering the “all ones” generator $\mathbf{1}$ and taking the rest
 416 of the generators from U according to the 0-bits in x (lines 6–7). The last
 417 part in lines 8–14 tackles the case where the two choices have equal length,
 418 so one of them is randomly chosen.

419 Let also note that, using the new generating set $\hat{U}$, the weight of any
 420 $x \in \mathbb{B}^m$, i.e., $|x| = |\text{Factorization}(x)|$, does not correspond anymore to the
 421 Hamming weight. Moreover, the maximum distance in the search space is
 422 now $\lceil m/2 \rceil$.

423 Finally, by considering the new generating set, its factorization algorithm
 424 and the induced weight function, the operations $\oplus, \ominus, \odot$ continue to work
 425 as previously described, though their semantic interpretation is now in line
 426 with the phenotypic space of the MDTWNPP binary partitions.

427 4.3. Search characteristics of the binary differential mutation in iMADEB

428 Here we analyze the implementation of the binary algebraic differential
 429 mutation provided in equation (6) for the newly introduced non-redundant
 430 representation.

Let describe the computation of the mutant $y_i \leftarrow x_i \oplus F \odot (x_{r_1} \ominus x_{r_2})$ by
 means of an illustrative example. Let consider $n = 8$ (thus $m = 7$ and v_8 is
 the reference vector), $F = 0.66$ and the following assignments for x_i, x_{r_1}, x_{r_2} :

$$\begin{aligned} x_i &= (0101010), \\ x_{r_1} &= (0010010), \\ x_{r_2} &= (1101110). \end{aligned}$$

We analyze the mutation equation from right to left, thus we start by
 observing that x_{r_1} and x_{r_2} represent the MDTWNPP partitions

$$\begin{aligned} (S_0^{x_{r_1}} = \{v_1, v_2, v_4, v_5, v_7, v_8\}, \quad S_1^{x_{r_1}} = \{v_3, v_6\}), \\ (S_0^{x_{r_2}} = \{v_3, v_7, v_8\}, \quad S_1^{x_{r_2}} = \{v_1, v_2, v_4, v_5, v_6\}). \end{aligned}$$

431 Their genotypic difference is $\delta = x_{r_1} \ominus x_{r_2} = x_{r_1} \vee x_{r_2} = (1111100)$ and,
 432 using Algorithm 2, is factorized as $\hat{U}_\delta = \{u_6, u_7, \mathbf{1}\}$. In fact, it is easy to see
 433 that $(S_0^{x_{r_1}}, S_1^{x_{r_1}})$ can be obtained from $(S_0^{x_{r_2}}, S_1^{x_{r_2}})$ by changing the subset
 434 of the MDTWNPP vectors corresponding to the generators in $\hat{U}_\delta$, i.e., the
 435 vectors v_6, v_7 and v_8 . Therefore, the weight of the difference bit-string is
 436 $|\delta| = |\hat{U}_\delta| = 3$.

437 Now, in order to compute the scalar multiplication $F \odot \delta = 0.66 \odot \delta$, we
 438 need to randomly select $\lceil 0.66 \cdot |\delta| \rceil = 2$ generators from $\hat{U}_\delta$. Let suppose we
 439 take the generators u_6 and u_7 from $\hat{U}_\delta$, then $0.66 \odot \delta = u_6 \vee u_7 = (0000011)$.

Finally, $y_i = x_i \oplus (F \odot \delta) = (0101010) \vee (0000011) = (0101001)$. Here, it is interesting to note that, in accordance with the generators in the decomposition of $F \odot \delta$, the last two bits of x_i are flipped. Moreover, let observe that x_i encodes the partition

$$(S_0^{x_i} = \{v_1, v_3, v_5, v_7, v_8\}, S_1^{x_i} = \{v_2, v_4, v_6\}),$$

while the mutant y_i represents the partition

$$(S_0^{y_i} = \{v_1, v_3, v_5, v_6, v_8\}, S_1^{y_i} = \{v_2, v_4, v_7\}).$$

440 As expected, $(S_0^{y_i}, S_1^{y_i})$ is obtained from $(S_0^{x_i}, S_1^{x_i})$ by changing the subset of
 441 the MDTWNPP vectors v_6 and v_7 .

442 In general, we have that the number of vectors which change subset in
 443 the partition represented by x_i is given by the weight of the scaled difference
 444 between x_{r_1} and x_{r_2} . Moreover, the vectors which are allowed to change
 445 subset in the partition corresponding to x_i are those which appear in different
 446 subsets in the partitions represented by x_{r_1} and x_{r_2} .

447 Furthermore, it is worthwhile to note that most of the binary crossovers
 448 in the literature are somehow special cases of our binary differential muta-
 449 tion. Let think for example to the very popular uniform crossover, one-point
 450 crossover or the more general k-points crossover (Pavai and Geetha, 2016).
 451 All of them, when applied to two generic bit-strings x and y , produce an
 452 offspring z such that its j -th bit $z(j)$ is equal to either $x(j)$ or $y(j)$. It is easy
 453 to see that the computation of an offspring with such a property can be easily
 454 reproduced in the algebraic framework as $z = x \oplus F \odot (y \ominus x)$ and by setting
 455 $F \in [0, 1]$. Therefore, binary crossovers are special cases of our differential
 456 mutation. This motivates the absence of a crossover operator in iMADEB,
 457 which in turn is the reason of why we have chosen the DE mutation variant
 458 where the current individual x_i is used as base solution to be mutated (Storn
 459 and Price, 1997).

460 5. Lévy flight adaptation

461 The exploration strength of iMADEB is regulated by the scale factor
 462 parameter F of equation (6). In the following, after analyzing the impact of

463 F on the search, we introduce a self-adaptive scheme built on the basis of
 464 the Lévy flight concept (Viswanathan et al., 1999).

465 During iMADEB evolution it may happen that the population reaches
 466 the consensus on a generic bit j – i.e., all individuals have their j -th bit set
 467 to the same value –, hence the decomposition of a binary difference between
 468 any pair of population individuals provably cannot include the generator
 469 corresponding to bit j . Therefore, by setting the scalar factor $F \in (0, 1]$
 470 – as usual in the numeric DE literature (Storn and Price, 1997) – the j -
 471 th bit of the base individual x_i cannot be flipped anymore by the binary
 472 differential mutation. From one hand, this aspect allows the search to focus
 473 an a “consensus subspace” learned during the evolution but, on the other
 474 hand, it may bring to a premature convergence to sub-optimal solutions.

475 Fortunately, the scalar multiplication by a scale factor $F > 1$ extends
 476 the binary difference $x_{r_1} \ominus x_{r_2}$ by introducing generators corresponding to its
 477 0-bits, thus the binary differential mutation can now flip a bit value of x_i ,
 478 even if the population has reached consensus on it. Therefore, setting $F > 1$
 479 may allow to escape stagnation states, but it may result in random search
 480 behaviours if a large value is used.

481 What is required is to regulate F dynamically during the search in such
 482 a way that: most of the times F is set to small values in order to make the
 483 search focus on the nearby areas of the current population, while occasion-
 484 ally larger values are used to prevent premature convergence to sub-optimal
 485 regions. It is interesting to notice that this is the typical motion pattern of
 486 the Lévy flight processes, which have been observed in many natural and
 487 artificial systems (Viswanathan et al., 1999; Iacca et al., 2020; Tomassini,
 488 2016). Lévy flight processes are based on the Lévy distribution, whose den-
 489 sity function decays in asymptotic power-law form. Although the Lévy law
 490 covers a wide class of distributions, in this work we adopt the simplified case,
 491 as used also in (Tomassini, 2016), where the scale factor values are drawn
 492 from a power-law probability distribution.

493 The adaptation mechanism of iMADEB is also based on the popular jDE
 494 scheme (Brest et al., 2006) – as used in our previous proposal (Santucci et al.,
 495 2019) – and it works as follows. Every population individual x_i maintains its
 496 own F_i value. The mutant y_i is computed using a scale factor F_{trial} which:
 497 with probability 0.9 is set to F_i , otherwise it is randomly generated according
 498 to the power-law distribution with density $\phi(F)$, as defined below. The
 499 mutant y_i undergoes local search giving rise to the offspring individual z_i that,
 500 if fitter, besides replacing x_i in the population, it also updates F_i to F_{trial} .

The power-law density $\phi(F)$ is set such that the sampled scale factors are larger than $F_{\min}$ and no upper bound is given, i.e., $F \in [F_{\min}, +\infty)$. Formally,

$$\phi(F) = \frac{\alpha - 1}{F_{\min}^{1-\alpha}} \cdot F^{-\alpha}, \quad (7)$$

501 where: the first factor is a normalization factor that depends on the value
 502 of $F_{\min}$ which is fixed to 0.1 as in (Brest et al., 2006), while $\alpha > 1$ is a
 503 parameter that regulates how quickly the probability density fades away when
 504 F increases.

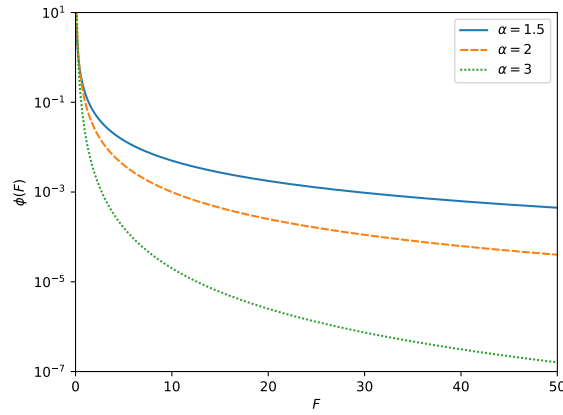


Figure 3: Power-law density (in log-scale) for $\alpha \in \{1.5, 2, 3\}$

505 The behaviour of $\phi(F)$, for $\alpha \in \{1.5, 2, 3\}$, is depicted in Figure 3, which
 506 shows that the probability density is larger for small values of F close to 0.1
 507 but, importantly, remains positive for larger F values, thus allowing iMADEB
 508 to occasionally generate mutant solutions far away from the current popula-
 509 tion.

510 6. Variable Neighborhood Descent

511 In iMADEB, a parameter $p_{LS} \in [0, 1]$ regulates the probability that a mu-
 512 tant individual undergoes a local search phase, implemented using a variable
 513 neighborhood descent scheme. Besides the probabilistic application strategy,
 514 other differences with respect to our previous proposal (Santucci et al., 2019)
 515 are: a new neighborhood definition and the best-improvement exploration
 516 scheme.

517 The `VariableNeighborhoodDescent` procedure takes in input a mutant
 518 $y \in \mathbb{B}^{n-1}$ and returns an (hopefully) improved solution $z \in \mathbb{B}^{n-1}$ which is a
 519 common local minimum with respect to the two neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_{1.5}$,
 520 defined as follows.

521 $\mathcal{N}_1$ is the classic *1-change* neighborhood. Given $y \in \mathbb{B}^{n-1}$, which encodes
 522 the binary partition (S_0, S_1) (as defined in Section 3), $\mathcal{N}_1(y)$ is the set of par-
 523 titions that can be obtained from (S_0, S_1) by changing the subset of exactly
 524 one MDTWNPP vector, which moves either from S_0 to S_1 or in the opposite
 525 direction. Formally, by considering the generating set $\hat{U}$ defined in Section 4,
 526 $\mathcal{N}_1(y) = \{y \vee u : u \in \hat{U}\}$. Hence, $|\mathcal{N}_1(y)| = n$, for any $y \in \mathbb{B}^{n-1}$.

527 $\mathcal{N}_{1.5}$ is a *restricted 2-change* neighborhood, defined in a similar way as
 528 in (Rodriguez et al., 2017). Given the current solution $y \in \mathbb{B}^{n-1}$ and its
 529 corresponding partition (S_0, S_1) , then $\mathcal{N}_{1.5}(y)$ contains the partitions which
 530 can be obtained from (S_0, S_1) by simultaneously changing the subset of two
 531 MDTWNPP vectors v and w such that: v belongs to the larger subset be-
 532 tween S_0 and S_1 , while w is selected as the most similar vector to v in the
 533 other subset, in terms of the L^∞ distance defined as in equation (1). Clearly,
 534 also $\mathcal{N}_{1.5}(y)$ can be algebraically expressed as y xored with two suitable gen-
 535 erators from $\hat{U}$.

536 $\mathcal{N}_{1.5}$ replaces the full 2-change neighborhood $\mathcal{N}_2$ used in our previous
 537 proposal (Santucci et al., 2019). This choice is motivated by the fact that
 538 the size of $\mathcal{N}_{1.5}$ is linear in n , while that of $\mathcal{N}_2$ is quadratic. In fact, $|\mathcal{N}_{1.5}(y)|$
 539 depends from the current solution y and: it is 0 when one of the partition
 540 subsets of y is empty, otherwise it is equal to the size of largest partition
 541 subset. Hence, $n/2 \leq |\mathcal{N}_{1.5}(y)| < n$ for any $y \in \mathbb{B}^{n-1}$ such that $y \neq \mathbf{0}$.

542 Moreover, the 2-change moves allowed by $\mathcal{N}_{1.5}(y)$ are those which modify
 543 the objective value of y as little as possible, thus allowing a smoother explo-
 544 ration of the search landscape. In fact, note that the L^∞ distance, used to
 545 select the pair of MDTWNPP vectors to swap, is used also in the objective
 546 function formulation given in equation (2). This is an important difference
 547 with respect to the restricted neighborhood proposed in (Rodriguez et al.,
 548 2017), where the Euclidean distance – unrelated with MDTWNPP objective
 549 function – is adopted.

550 The pseudocode of `VariableNeighborhoodDescent` is given in Algo-
 551 rithm 3, where it is possible to see that the two neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_{1.5}$
 552 are alternatively explored until no improving solution is obtained.

553 Since both neighborhoods have a linear size, we decided to use a more
 554 thorough best-improvement search, i.e., the neighborhoods are fully explored

Algorithm 3 Pseudocode of VariableNeighborhoodDescent

```
1: function VARIABLENEIGHBORHOODESCENT( $y \in \mathbb{B}^{n-1}$ )
2:   repeat ▷ Outer loop on both  $\mathcal{N}_1$  and  $\mathcal{N}_{1.5}$ 
3:      $y_{\text{old}} \leftarrow y$ 
4:     repeat ▷ Loop on  $\mathcal{N}_1$ 
5:        $x \leftarrow y$ 
6:        $y \leftarrow \arg \min_{z \in \mathcal{N}_1(y)} f(z)$ 
7:     until  $f(y) \geq f(x)$ 
8:      $y \leftarrow x$ 
9:     repeat ▷ Loop on  $\mathcal{N}_{1.5}$ 
10:       $x \leftarrow y$ 
11:       $y \leftarrow \arg \min_{z \in \mathcal{N}_{1.5}(y)} f(z)$ 
12:    until  $f(y) \geq f(x)$ 
13:     $y \leftarrow x$ 
14:  until  $f(y) = f(y_{\text{old}})$ 
15:  return  $y$ 
16: end function
```

555 and the best neighbor is considered as trial solution (lines 6 and 11).

556 Finally, it is important to note that the evaluation of a neighbor is not
557 made from scratch, but incrementally with respect to the incumbent solution.
558 In fact, by maintaining the two partial subset sums of the current solution, it
559 is possible to calculate the objective value of a $\mathcal{N}_1$ neighbor by means of one
560 vector addition, one vector subtraction and one distance computation. All
561 these operations cost $\Theta(d)$ time. Therefore, every iteration of the loop in lines
562 4–7 costs $\Theta(nd)$ operations. Furthermore, for the $\mathcal{N}_{1.5}$, we can precompute –
563 at the beginning of an iMADEB execution – the distances among all the pairs
564 of MDTWNPP vectors, thus a neighbor evaluation can be done by means of:
565 two vector additions, two vector subtractions and one minimum computation
566 in order to find the closest vector in the other subset. This last operation
567 costs $\Theta(n)$ time. Therefore, every iteration of the loop in lines 9–12 costs
568 $\Theta(n \cdot \max\{n, d\})$ operations.

569 7. Experiments

570 In order to analyze iMADEB and assess its effectiveness, a number of
571 experiments have been held by considering commonly adopted benchmark
572 instances for the MDTWNPP.

573 The algorithm has been implemented in C++ and all the experiments
 574 have been carried out on a machine equipped with an Intel Xeon E312 clock-
 575 ing at 2.2 GHz, 16 GB of RAM and running Linux Ubuntu 18.04.

576 The iMADEB parameters have been experimentally tuned and analyzed
 577 as described in Section 7.1. Then, the calibrated iMADEB setting has been
 578 experimentally compared with the other state-of-the-art algorithms. This
 579 comparison is described and discussed in Section 7.2.

580 *7.1. Tuning and analysis of the iMADEB parameters*

581 iMADEB has three parameters to be set: the population size N , the Lévy
 582 flight parameter α , and the local search application probability p_{LS} . After a
 583 series of preliminary experiments, a discrete set of values has been selected
 584 for each parameter. Then, a full factorial experiment has been carried out for
 585 selecting the most effective setting and analyzing the robustness of iMADEB.
 586 For each parameter, the chosen values are:

- 587 • $N \in \{50, 100, 150, 200\}$,
- 588 • $\alpha \in \{1.5, 2, 3\}$,
- 589 • $p_{LS} \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$.

590 These $4 \times 3 \times 5 = 60$ settings of iMADEB have been experimented on a set
 591 of 16 benchmark instances: one instance for every n, d problem configuration,
 592 with $n \in \{20, 100, 300, 500\}$ and $d \in \{2, 5, 15, 20\}$. To avoid the over-tuning
 593 phenomenon, the tuning instances have been generated in such a way they
 594 are representative of the test instances, but different from them. Hence, a
 595 matrix of 500×20 numbers is randomly generated and sub-sampled for the
 596 different values of n and d , as done in (Kojić, 2010) for producing the test
 597 suite adopted in this work (see Section 7.2) and in all the previous works in
 598 the MDTWNPP literature (see Section 2).

599 Every iMADEB setting has been executed 25 times per instance with a
 600 computational budget of 240 seconds per execution. Therefore, 24000 ex-
 601 ecutions have been carried out for a total of 9600 hours of computational
 602 time.

The performance of each iMADEB setting S , on every instance i , is
 measured by the commonly adopted average relative percentage deviation

(ARPD) index, defined as

$$\text{ARPD}_i^S = \frac{1}{25} \sum_{j=1}^{25} \frac{S_i^j - \text{Best}_i}{\text{Best}_i} \times 100, \quad (8)$$

where S_i^j is the objective value obtained by the iMADEB setting S in its j -th run on the instance i , and Best_i is the best objective value achieved among all the performed executions on instance i .

First of all, we analyze the impact of any single parameter setting on the effectiveness of iMADEB. With this regard, in Figure 4 we provide three box-plot graphs – one for each parameter – which graphically summarize the ARPDs obtained varying each parameter value.

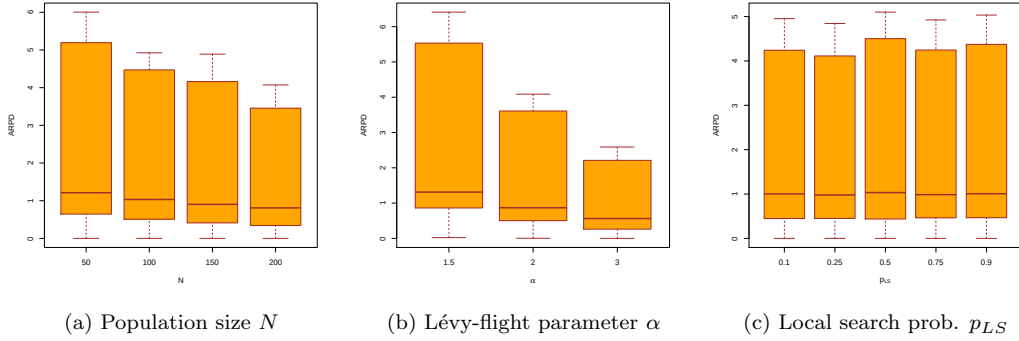


Figure 4: Box-plot graphs from the calibration of the three iMADEB parameters

Figure 4a shows that a population size of $N = 200$ is to be preferred, though its impact on the effectiveness of an iMADEB execution is not as large as it is for the setting $\alpha = 3$. In fact, Figure 4b clearly shows that the α parameter has an important role. By recalling the behaviour of the probability density shown in Figure 3, the top performances obtained with $\alpha = 3$ suggest that iMADEB prefers to intensify the search in the nearby of the current population individuals and, only very occasionally, exploring distant areas in the space. Conversely from the previous cases, Figure 4c shows that iMADEB executions are robust throughout different settings of the p_{LS} parameter.

In order to validate these considerations, we statistically analyzed the results presented in Figure 4 by means of the Kruskal-Wallis H test (Hollander et al., 2013). One test is performed for every parameter aiming at

understanding if the difference in the observed performances is statistically significant or not.

For the population size N , the statistical test returned a p-value of 0.01, thus confirming that a good setting for the population size is significant in order to obtain good performances. For the parameter α the significance is even stronger, since the returned p-value is smaller than 10^{-5} , while the very large p-value (0.99) obtained for p_{LS} confirms that the local search application probability does not impact too much the effectiveness of iMADEB.

Aiming to analyze the complete parameters configurations, in Table 1 we provide the top performing iMADEB settings, ordered by average rank, together with their overall ARPDs. The average rank of any setting S is computed by averaging the ranks obtained by S – among the 60 different settings – throughout all the 16 tuning instances.

Moreover, we carried out the Friedman statistical test (Hollander et al., 2013) which returned an almost zero p-value, thus indicating statistical differences among the 60 settings. Hence, a post-hoc analysis has been conducted by considering all the Friedman post-hoc procedures available in the statistical package `scikit-posthocs` (Terpilowski, 2019) and selecting the most discriminating one that, in our case, was the Siegel and Castellan test with the Benjamini/Hochberg p-value adjustment scheme (Hollander et al., 2013; Terpilowski, 2019). Therefore, in Table 1 we list the largest set of top performing settings which are not statistically different to each other, by considering a significance threshold of 0.05. Moreover, we also provide the post-hoc p-values of the pairwise comparisons between the best setting and the other ones.

From Table 1 it is possible to see that 15 settings, out of 60, do not show significant performance differences with respect to each other, thus indicating a good robustness of iMADEB. Moreover, it is interesting to observe that: all the 15 settings in Table 1 have $\alpha = 3$, while the top five settings have $N = 200$ and all the possible values for p_{LS} . These observations clearly confirm the previously discussed analyses. Let also note that the ARPDs in Table 1, though not being in a perfectly monotonic relationship with the average ranks, show a negligible variance – the largest is only 1.2 percentage points larger than the smallest –, thus further confirming the overall good robustness of iMADEB.

Finally, the best setting of parameters is ($N = 200, \alpha = 3, p_{LS} = 0.9$), which reached the lowest average rank of 5.06. Therefore, this is the setting used for the experimental comparison discussed in Section 7.2.

Table 1: The 15 most performing iMADEB settings ordered by average rank

Setting			Average	Overall	Post-hoc
N	α	p_{LS}	Rank	ARPD	p-value
200	3.0	0.90	5.06	1.84	best
200	3.0	0.25	5.09	1.74	0.41
200	3.0	0.10	5.09	1.60	0.41
200	3.0	0.75	5.50	1.85	0.40
200	3.0	0.50	5.62	1.67	0.40
150	3.0	0.10	7.69	1.94	0.31
150	3.0	0.25	8.25	1.92	0.29
150	3.0	0.50	9.88	2.27	0.22
150	3.0	0.90	9.97	2.31	0.21
100	3.0	0.50	11.16	1.63	0.17
150	3.0	0.75	11.34	2.18	0.16
100	3.0	0.25	12.62	2.80	0.12
100	3.0	0.10	12.97	1.97	0.11
100	3.0	0.75	14.03	2.47	0.09
100	3.0	0.90	14.41	2.39	0.08

661 7.2. Experimental comparison with the state-of-the-art algorithms

662 In order to compare the effectiveness of iMADEB with respect to the
663 other state-of-the-art algorithms for the MDTWNPP, the set of benchmark
664 instances proposed in (Kojić, 2010), and adopted in all the other works in
665 the MDTWNPP literature (see Section 2), is considered. The benchmark
666 suite is formed by a total of 210 instances: five for any problem configuration
667 n, d such that $n \in \{50, 100, 200, 300, 400, 500\}$ and $d \in \{2, 3, 4, 5, 10, 15, 20\}$.

668 iMADEB has been executed using the setting of parameters identified in
669 Section 7.1 and it is compared with the two state-of-the-art algorithms to
670 date: MADEB (Santucci et al., 2019) and GRASP+ePR (Rodriguez et al.,
671 2017). In order to perform a fair comparison, all the three algorithms have
672 been executed on the same machine and using the same budget of compu-
673 tational time. Moreover, the executable code of GRASP+ePR has been got
674 from the website provided by the authors (<https://sci2s.ugr.es/MDTWNPP>),
675 and both GRASP+ePR and MADEB have been run with the parameters set-
676 tings suggested in, respectively, (Santucci et al., 2019) and (Rodriguez et al.,
677 2017).

Each algorithm has been executed 25 times per instance with a budget of 600 seconds per execution. Therefore, 15750 executions have been carried out for a total of 2625 hours of computational time.

For each algorithm, we have computed its ARPD measures which are also used to rank the algorithms on every instance, then the ranks are averaged and shown in Table 2 grouped by n . Table 2 also provides the number of instances where an algorithm obtained the best objective value among all the executions of every competitors. The best results are indicated in bold, while the last line provides the overall average ranks and the total number of instances where any algorithm obtained the best solution.

Table 2: Average ranks and number of best solutions obtained

n	<u>Average Rank</u>			<u>No. Best Solutions</u>		
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	GRASP +ePR
50	1.74	1.57	2.69	29	6	7
100	1.26	1.77	2.97	32	4	2
200	1.20	1.80	3.00	33	3	0
300	1.17	1.83	3.00	34	1	0
400	1.06	1.94	3.00	35	0	0
500	1.11	1.89	3.00	35	1	0
Overall	1.26	1.80	2.94	198	15	9

Overall, Table 2 clearly shows that iMADEB outperformed its competitors in terms of both average and peak results. In fact, its overall average rank is 1.26 – very close to the optimal ideal value of 1 –, while it obtained the best solution on 198 out of 210 instances, i.e., more than the 94% of the benchmark suite.

Moreover, it is interesting to note that iMADEB consistently outperformed GRASP+ePR, both in terms of average ranks and number of best solutions, across all the different values of n . The same is true also when compared with respect to its predecessor MADEB that, anyway, obtained a slightly better average rank in the case $n = 50$. However note that, in the same group of instances, iMADEB obtained a consistently larger number of best solutions, thus indicating that, when n is relatively small, iMADEB is not as robust as for larger instances, but anyway able to obtain the best peak performances.

702 In order to better compare the effectiveness of the three algorithms, in
 703 Figure 5 we provide the box-plot graphs which graphically summarize the
 704 ARPDs obtained by varying both the instance parameters n (Figure 5a) and
 705 d (Figure 5b). For the sake of presentation, the logarithm of the ARPD
 706 values is considered.

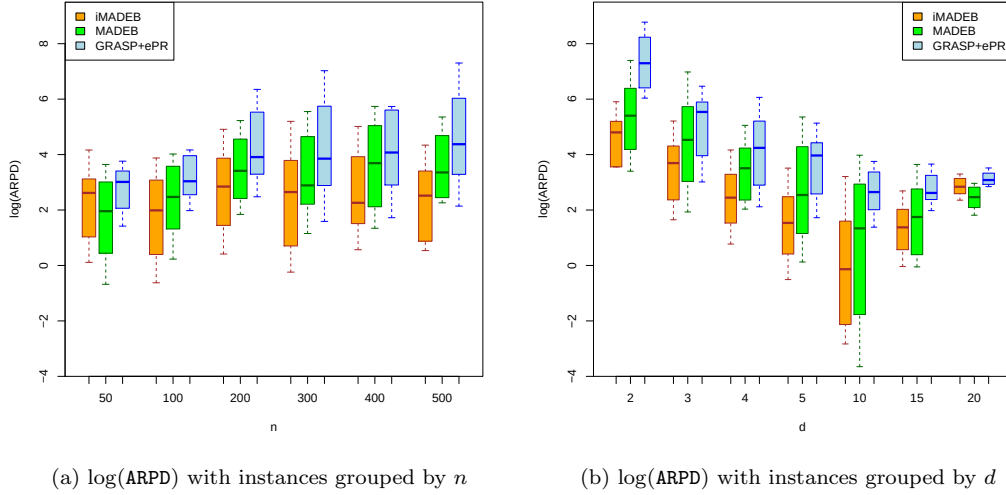


Figure 5: Box-plot graphs of the $\log(\text{ARPD})$ measure with instances grouped by n and d

707 Both box-plots show that iMADEB is considerably more effective than
 708 GRASP+ePR. Moreover, Figure 5a largely confirms all the indications given
 709 by the average ranks, while Figure 5b shows that iMADEB obtained better
 710 median results for every value of d , except the case $d = 20$ where it is
 711 outperformed by its predecessor MADEB.

712 In order to validate these considerations, we statistically analyzed the
 713 comparisons by running two pairwise Wilcoxon tests (Hollander et al., 2013)
 714 – iMADEB vs MADEB and iMADEB vs GRASP+ePR – on every group of
 715 instances aggregated as in Figure 5.

716 Grouping the instances by n , iMADEB significantly outperformed both
 717 competitors when $n \geq 100$. In these cases, the largest p-value observed
 718 is smaller than 0.003. Conversely, for $n = 50$ – where MADEB obtained
 719 slightly better results – the differences in performances are not statistically
 720 significant. In fact, the p-value of the comparison with MADEB has the very
 721 large p-value of 0.59, while that with respect to GRASP+ePR is 0.06.

722 Grouping the instances by d , iMADEB significantly outperformed both
723 competitors when $d \leq 10$. In these cases, the largest p-value is smaller than
724 $5 \cdot 10^{-4}$. When $d = 15$, iMADEB significantly outperforms GRASP+ePR
725 (with a p-value of 0.002), while it is statistically indistinguishable from
726 MADEB (with a p-value of 0.50). The only case where iMADEB is signifi-
727 cantly outperformed is in the group of instances with $d = 20$, where MADEB
728 obtained a better median ARPD and the Wilcoxon test returned a p-value
729 of around 10^{-4} .

730 Importantly, two additional Wilcoxon tests have been also conducted by
731 considering the whole set of instances: iMADEB significantly outperformed
732 both MADEB and GRASP+ePR with p-values very close to zero.

733 For the sake of completeness, in Table 3 we provide, for all the 210
734 instances, the average and best objective values obtained by the three al-
735 gorithms considered in our experimentation. For each instance it is also
736 reported the previously best known objective value (by considering all the
737 works described in Section 2). Best results are indicated in bold, while the ob-
738 jective values which improves the previously best known solutions are marked
739 with an asterisk.

740 In particular, it is interesting to observe this last datum: iMADEB ob-
741 tained 145 new best known solutions, i.e., around the 69% of the benchmark
742 suite. Moreover, few new best known solutions have been obtained by our
743 new executions of MADEB (5) and GRASP+ePR (2). In conclusion, Table 3
744 provides a comprehensive perspective of the state-of-the-art results for the
745 most used MDTWNPP benchmark suite at the time of writing.

746 8. Conclusion and Future Work

747 In this work, we have proposed a new memetic algorithm for the MultiDi-
748 mensional Two-Way Number Partitioning Problem (MDTWNPP), namely
749 iMADEB, which adopts an algebraic differential mutation operator for ex-
750 ploring the search space and providing new seed solutions to a local search
751 phase implemented as a variable neighborhood descent procedure.

752 Our proposal is motivated by a critical analysis of the MDTWNPP liter-
753 ature. In fact, all the previously proposed meta-heuristics adopt a redundant
754 representation scheme for the solutions and do not consider the intrinsic char-
755 acteristics of the MDTWNPP objective function in the design of the local
756 search neighborhoods.

Table 3: Detailed experimental results on all the 210 instances

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	IMADEV	MADEV	GRASP +ePR	IMADEV	MADEV	GRASP +ePR	
50_2a	0.61	0.48	0.95	0.45	0.45	0.45	0.45
50_2b	1.42	3.29	6.54	0.26 *	0.26	3.09	1.62
50_2c	1.31	2.88	6.10	1.17 *	1.17	3.09	2.48
50_2d	1.44	1.48	2.92	1.34 *	1.34	1.34	1.34
50_2e	3.23	4.32	6.78	0.62 *	2.63	2.94	2.94
50_3a	277.12	288.32	292.10	236.42 *	282.22	285.27	283.56
50_3b	302.33	307.21	345.91	287.40 *	302.63	308.74	305.68
50_3c	136.08	142.17	147.21	118.52 *	139.88	142.94	139.88
50_3d	93.60	99.31	104.22	81.14 *	93.36	96.41	96.41
50_3e	205.19	212.56	216.80	187.39 *	208.75	211.79	198.85
50_4a	900.71	905.80	941.98	894.51	900.61	903.66	783.05
50_4b	1217.30	1221.82	1240.66	1108.17	1216.49	1222.59	1017.28
50_4c	442.24	453.68	456.52	421.50 *	448.95	452.01	452.01
50_4d	1011.45	1019.49	1021.80	1000.26	1015.52	1018.58	987.27
50_4e	1193.75	1204.30	1215.70	970.11 *	1202.62	1202.62	1187.27
50_5a	920.32	922.81	1283.06	917.01 *	920.06	920.06	917.01
50_5b	2609.12	2670.97	3186.74	2600.99 *	2613.19	2616.25	2607.09
50_5c	1394.91	1402.28	2471.71	1386.57 *	1398.77	1410.98	1395.72
50_5d	2269.08	2271.98	2271.20	2260.52	2269.69	2280.69	2183.62
50_5e	3959.34	4404.84	4608.16	3649.46 *	3655.57	4411.06	3652.52
50_10a	16273.64	16175.67	16423.06	16170.50	16173.50	16173.52	15722.29
50_10b	19782.24	19553.17	20356.29	19548.10	19551.20	19557.26	19548.10
50_10c	17117.30	17392.40	17687.80	14131.60	14134.70	14134.69	14135.54
50_10d	16960.89	14924.10	17121.38	14918.90	14918.90	14921.98	14915.87
50_10e	16834.22	15364.59	16504.30	15356.80	15359.90	15356.80	14527.38
50_15a	38593.84	33523.49	34942.80	33208.00	33208.00	3321.107	30728.55
50_15b	39546.42	36385.14	36829.60	33240.00	33240.00	33240.00	33240.00
50_15c	40996.84	32470.97	35748.85	29920.30	29456.90	29456.85	28736.38
50_15d	35572.00	25222.35	28584.60	21649.80	21652.80	21655.89	20356.84
50_15e	37501.69	34565.42	35754.57	25043.00	25180.70	25013.28	23893.88
50_20a	61084.24	56919.47	58485.54	55685.60	52826.30	52826.34	50647.84
50_20b	63479.59	55173.79	58729.66	53898.60	51917.90	51917.90	50832.38
50_20c	63775.68	54141.04	57999.14	50560.90	50560.90	50560.86	50560.86
50_20d	61134.74	52983.35	56785.88	53965.00	51538.57	51538.57	50850.88
50_20e	61339.02	51239.31	57688.77	48281.50	48281.50	48281.50	47829.86
200_2a	0.25	0.93	11.44	0.03 *	0.45	3.77	0.44
200_2b	0.55	0.65	2.24	0.36 *	0.56	0.86	0.56
200_2c	0.53	0.63	2.57	0.22 *	0.30	0.70	0.20
200_2d	0.62	0.67	3.67	0.23 *	0.43	0.67	0.24
200_2e	0.43	0.44	1.78	0.27 *	0.27	0.45	0.44
200_3a	96.46	172.97	188.88	66.79 *	163.06	175.27	85.10
200_3b	86.12	129.95	140.51	51.65 *	125.37	131.48	71.38
200_3c	1.57	2.13	5.13	1.51 *	1.52	1.52	1.52
200_3d	35.01	104.78	118.05	25.09 *	99.30	111.50	47.04
200_3e	7.66	103.45	201.57	43.90 *	90.02	191.32	83.57
200_4a	481.98	509.55	1160.65	460.72 *	503.44	1144.04	494.29
200_4b	1049.55	1167.65	1179.58	890.57 *	1157.73	1169.94	914.98
200_4c	2.49	2.82	5.55	1.51 *	1.52	3.05	3.05
200_4d	665.40	994.38	1065.87	588.22 *	987.61	1015.07	685.87
200_4e	458.45	501.92	1232.18	416.18 *	489.70	1223.65	486.65
200_5a	1614.04	1964.98	1943.95	1202.05	1897.51	1909.71	1217.32
200_5b	1831.75	1961.72	2024.69	1754.33 *	1855.15	1876.53	1852.11
200_5c	3.00	3.13	6.29	1.52 *	3.05	3.05	3.05
200_5d	2009.86	2119.54	2613.65	1885.14 *	2106.26	2369.47	2112.36
200_5e	1293.18	2935.78	3784.08	1185.27 *	2038.66	3777.29	1212.74
200_10a	16357.65	16445.07	16566.46	16347.60	16362.80	16505.91	16347.58
200_10b	15414.13	17434.35	17795.58	15128.50	15456.60	17782.27	15128.47
200_10c	12778.96	12791.56	16159.14	12767.50 *	12782.70	16149.50	12782.73
200_10d	14195.15	17569.26	18690.38	13101.80 *	17887.20	18696.25	13107.86
200_10e	17115.53	17793.69	17801.03	15905.60	17787.90	17790.89	15896.50
200_15a	29295.89	29398.32	31794.17	28373.90 *	29382.30	29388.40	28389.13
200_15b	30534.78	30474.24	32326.28	30467.50	30458.40 *	30473.64	30467.53
200_15c	30373.10	30473.65	30881.96	30083.70 *	30465.30	30470.06	30463.49
200_15d	24055.90	23563.88	31658.01	22882.70 *	22825.80	32831.88	22834.94
200_15e	29235.59	29024.40	32404.98	28002.50	28062.50	32868.64	27483.37
200_20a	47708.70	46793.39	51427.54	40931.60	42045.50	42606.76	40385.54
200_20b	48847.39	48502.36	51358.95	41671.70 *	42990.30	46268.19	42311.89
200_20c	49039.63	48137.58	50790.92	36926.60 *	41999.80	45571.04	36939.80
200_20d	49267.76	49608.93	50697.35	39539.10 *	41186.90	41168.59	39539.13
200_20e	49504.85	48439.67	51614.55	41777.30	37156.10 *	41777.28	39408.54
400_2a	0.07	0.13	7.01	0.07 *	0.08	0.09	0.09
400_2b	0.27	0.48	0.57	0.12 *	0.34	0.59	0.29
400_2c	0.33	0.55	2.76	0.04 *	0.20	0.88	0.22
400_2d	0.35	0.67	3.91	0.04 *	0.63	1.04	0.41
400_2e	0.29	0.37	2.63	0.12 *	0.27	0.41	0.10
400_3a	60.41	163.36	190.25	15.21 *	152.37	170.69	72.35
400_3b	60.03	132.87	155.63	11.32 *	120.97	136.23	121.23
400_3c	11.93	47.50	241.23	1.48 *	39.10	214.74	45.21
400_3d	165.93	206.58	231.99	108.05 *	198.57	221.22	127.75
400_3e	75.70	146.91	209.79	51.21 *	133.64	170.26	95.18
400_4a	440.72	482.69	1154.91	408.55 *	471.40	1137.94	480.55
400_4b	486.03	536.47	682.44	366.06 *	522.89	550.36	541.21
400_4c	440.86	874.37	960.68	383.02 *	865.52	886.88	437.05
400_4d	548.54	789.44	898.03	264.88 *	779.46	872.84	794.72
400_4e	629.24	688.88	913.39	572.66 *	856.98	887.50	622.38
400_5a	1098.52	1798.36	1910.99	1041.83 *	1191.39	1902.07	1188.32
400_5b	1423.52	1452.66	1471.49	1350.88 *	1445.49	1460.74	1421.07
400_5c	885.53	899.77	916.62	857.61 *	895.64	907.85	802.13
400_5d	1576.54	2467.55	2499.68	1438.67 *	2450.62	2484.18	1646.22
400_5e	1567.76	1586.33	1613.46	1527.90	1574.88	1595.03	1465.10
400_10a	14117.06	14547.91	15203.17	13791.70	14274.80	14809.11	1701.83
400_10b	14812.86	17043.77	18757.56	14526.10	17029.99	17908.98	10827.98
400_10c	11671.89	15899.55	17115.57	10778.50	14503.10	15672.45	9202.61
400_10d	13712.90	13722.33	16215.57	13699.10	13706.19	15176.29	8212.93
400_10e	11547.89	16142.74	16225.97	11528.90 *	16135.10	16147.33	11562.44
400_15a	30206.00	30977.06	33196.13	29097.60 *	30440.80	32383.93	29134.27
400_15b	27782.30	28070.56	32528.68	25859.60	28056.50	28068.73	21278.92
400_15c	20841.13	20278.13	26215.57	20779.01 *	20781.13	26201.21	20812.90
400_15d	25904.86	29986.69	33462.53	22226.10	29973.40	29997.82	16172.37
400_15e	18272.13	30700.24	30927.16	18214.10 *	30111.40	30724.97	28423.13
400_20a	44900.56	45803.69	48755.62	40907.90	41937.70	45628.38	28134.23
400_20b	44106.86	44145.43	48047.22	40322.20	40325.30	43124.38	37938.93
400_20c	45280.90	46655.78	50454.56	41868.70	44834.60	44988.25	32762.87
400_20d	44787.21	44002.46	48454.07	41297.60	41887.50	41396.68	36726.27
400_20e	44206.29	42089.62	48320.45	38942.30 *	39541.10	39535.05	39541.10
500_2a	0.12	0.33	2.04	0.11 *	0.12	1.03	0.12
500_2b	0.28	0.51	1.76	0.10 *	0.18	0.57	0.47
500_2c	0.16	0.43	1.91	0.04 *	0.20	1.19	0.58
500_2d	0.20	0.25	2.28	0.18 *	0.20	0.41	0.19
500_2e	0.12	0.50	2.23	0.10 *	0.13	0.92	0.20
500_3a	1.52	1.52	5.74	1.52 *	1.52	1.52	1.52
500_3b	26.74	103.59	132.52	7.56 *	87.23	120.79	56.23
500_3c	33.71	92.21	181.89	10.45 *	84.04	138.96	68.88
500_3d	1.68	1.83	7.02	1.52 *	1.52	1.53	1.52
500_3e	1.52	1.53	5.98	1.51 *	1.52	1.53	1.52
500_4a	1.83	2.52	7.32	1.52 *	1.52 *	4.57	1.5

757 Therefore, in order to bridge this gap, iMADEB has been designed along
758 the following lines:

- 759 • a non-redundant binary representation for the MDTWNPP;
- 760 • an algebraic modeling for the new genotypic space;
- 761 • a self-adaptive mechanism, built on the basis of the Lévy flight concept,
762 for regulating the exploration-exploitation balance of the search;
- 763 • a restricted neighborhood which allows a smoother local exploration of
764 the space.

765 All these aspects are to be considered novelties with respect to previous
766 proposals.

767 Experiments have been held in order to analyze iMADEB robustness and
768 to compare its effectiveness with respect to the other state-of-the-art algo-
769 rithms. Regarding robustness, though iMADEB has three parameters to be
770 set, the experimental study carried out provides clear and robust indications
771 for the practitioners that need to choose an iMADEB setting. Most impor-
772 tantly, the experimental comparison with the previously proposed approaches
773 clearly show that iMADEB can be considered the new state-of-the-art algo-
774 rithm for the MDTWNPP, both in terms of average and peak results. More-
775 over, comprehensive experimental data are also provided in order to facilitate
776 comparisons.

777 Future studies may involve different lines. First of all, it is interesting to
778 study which features make an MDTWNPP instance difficult or easy to solve.
779 Moreover, the proposed approach can be generalized both to the multiway
780 variant of the MDTWNPP and to other partitioning problem such as, for
781 example, the graph partitioning problems. Another interesting line of re-
782 search is to study the novel algebraic method here proposed for other binary
783 optimization problems. Finally, the Lévy flight approach can be extended
784 also to other scenarios where an exploration-exploitation balance is required
785 in order to automatically regulate the focus of the search.

786 References

- 787 Baiocchi, M., Milani, A., Santucci, V., 2018. Learning bayesian networks
788 with algebraic differential evolution, in: Proc. of 15th Int. Conf. on Par-
789 allel Problem Solving from Nature – PPSN XV, Springer International
790 Publishing, Cham. pp. 436–448.

- 791 Baioletti, M., Milani, A., Santucci, V., 2020. Variable neighborhood algebraic
792 differential evolution: An application to the linear ordering problem with
793 cumulative costs. *Information Sciences* 507, 37–52.
- 794 Bi, Y., Srinivasan, D., Lu, X., Sun, Z., Zeng, W., 2014. Type-2 fuzzy multi-
795 intersection traffic signal control with differential evolution optimization.
796 *Expert Systems with Applications* 41, 7338–7349.
- 797 Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V., 2006. Self-
798 adapting control parameters in differential evolution: A comparative study
799 on numerical benchmark problems. *IEEE Transactions on Evolutionary*
800 *Computation* 10, 646–657.
- 801 Corus, D., Oliveto, P.S., Yazdani, D., 2018. Artificial immune systems can
802 find arbitrarily good approximations for the np-hard partition problem, in:
803 *Proc. of 15th Int. Conf. on Parallel Problem Solving from Nature – PPSN*
804 *XV*, pp. 16–28.
- 805 Cuevas, E., Zaldivar, D., Pérez-Cisneros, M., 2010. A novel multi-threshold
806 segmentation approach based on differential evolution optimization. *Ex-*
807 *pert Systems with Applications* 37, 5265–5271.
- 808 Faria, A.F., de Souza, S.R., de Sá, E.M., 2021. A mixed-integer linear pro-
809 gramming model to solve the multidimensional multi-way number parti-
810 tioning problem. *Computers & Operations Research* 127, 105133.
- 811 Hacibeyoglu, M., Alaykiran, K., Acilar, A.M., Tongur, V., Ulker, E., 2018.
812 A comparative analysis of metaheuristic approaches for multidimensional
813 two-way number partitioning problem. *Arabian Journal for Science and*
814 *Engineering* 43, 7499–7520.
- 815 Hacibeyoglu, M., Tongur, V., Alaykiran, K., 2014. Solving the bi-dimensional
816 two-way number partitioning problem with heuristic algorithms, in: *2014*
817 *IEEE 8th International Conference on Application of Information and*
818 *Communication Technologies (AICT)*, IEEE. pp. 1–5.
- 819 Hollander, M., Wolfe, D.A., Chicken, E., 2013. Nonparametric statistical
820 methods. volume 751. John Wiley & Sons.

- 821 Iacca, G., dos Santos Junior, V.C., de Melo, V.V., 2020. An improved jaya
822 optimization algorithm with lévy flight. *Expert Systems with Applications*
823 165, 113902.
- 824 Karmarker, N., Karp, R.M., 1983. The Differencing Method of Set Parti-
825 tioning. Technical Report. USA.
- 826 Karp, R.M., 1972. Reducibility among combinatorial problems, in: *Com-
827 plexity of computer computations*. Springer, pp. 85–103.
- 828 Kojić, J., 2010. Integer linear programming model for multidimensional two-
829 way number partitioning problem. *Computers & Mathematics with Ap-
830 plications* 60, 2302–2308.
- 831 Kratica, J., Kojić, J., Savić, A., 2014. Two metaheuristic approaches for solv-
832 ing multidimensional two-way number partitioning problem. *Computers
833 & Operations Research* 46, 59–68.
- 834 Mertens, S., 2006. The easiest hard problem: Number partitioning. *Compu-
835 tational Complexity and Statistical Physics* 125, 125–139.
- 836 Mladenović, N., Hansen, P., 1997. Variable neighborhood search. *Computers
837 & Operations Research* 24, 1097–1100.
- 838 Moscato, P., Cotta, C., 2003. A gentle introduction to memetic algorithms,
839 in: *Handbook of metaheuristics*. Springer, pp. 105–144.
- 840 Moscato, P., Cotta, C., 2019. An accelerated introduction to memetic algo-
841 rithms, in: *Handbook of metaheuristics*. Springer, pp. 275–309.
- 842 Moscato, P., Cotta, C., Mendes, A., 2004. Memetic algorithms, in: *New
843 optimization techniques in engineering*. Springer, pp. 53–85.
- 844 Pavai, G., Geetha, T.V., 2016. A survey on crossover operators. *ACM
845 Computing Surveys* 49, 1–43.
- 846 Penas, D., Banga, J., González, P., Doallo, R., 2015. Enhanced parallel differ-
847 ential evolution algorithm for problems in computational systems biology.
848 *Applied Soft Computing* 33, 86–99.

- 849 Piotrowski, A.P., 2013. Adaptive memetic differential evolution with global
850 and local neighborhood-based mutation operators. *Information Sciences*
851 241, 164–194.
- 852 Pop, P.C., Matei, O., 2013a. A genetic algorithm approach for the multidimensional two-way number partitioning problem, in: *Proc. of International Conference on Learning and Intelligent Optimization*, Springer. pp. 81–86.
- 853
854
- 855 Pop, P.C., Matei, O., 2013b. A memetic algorithm approach for solving
856 the multidimensional multi-way number partitioning problem. *Applied Mathematical Modelling* 37, 9191–9202.
857
- 858 Price, K., Storn, R.M., Lampinen, J.A., 2006. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media.
- 859
- 860 Rodriguez, F.J., Glover, F., García-Martínez, C., Martí, R., Lozano, M.,
861 2017. Grasp with exterior path-relinking and restricted local search for
862 the multidimensional two-way number partitioning problem. *Computers & Operations Research* 78, 243–254.
863
- 864 Santucci, V., Baioletti, M., Di Bari, G., Milani, A., 2019. A binary algebraic differential evolution for the multidimensional two-way number partitioning problem, in: *Proc. of the 19th European Conf. on Evolutionary Computation in Combinatorial Optimization – EvoCOP 2019*, Springer International Publishing, Cham. pp. 17–32.
- 865
866
867
868
- 869 Santucci, V., Baioletti, M., Milani, A., 2016. Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. *IEEE Transactions on Evolutionary Computation* 20, 682–694.
870
871
872
- 873 Santucci, V., Baioletti, M., Milani, A., 2020. An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization. *Swarm and Evolutionary Computation* 55, 100673.
874
875
- 876 Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11, 341–359.
877
878

- 879 Tanabe, R., Fukunaga, A., 2013. Success-history based parameter adaptation
880 for differential evolution, in: Proc. of 2013 IEEE Congress on Evolutionary
881 Computation – CEC 2013, IEEE. pp. 71–78.
- 882 Terpilowski, M.A., 2019. scikit-posthocs: Pairwise multiple comparison tests
883 in python. *Journal of Open Source Software* 4, 1169.
- 884 Tomassini, M., 2016. Lévy flights in neutral fitness landscapes. *Physica A:
885 Statistical Mechanics and its Applications* 448, 163–171.
- 886 Tsafarakis, S., Zervoudakis, K., Andronikidis, A., Altsitsiadis, E., 2020.
887 Fuzzy self-tuning differential evolution for optimal product line design.
888 *European Journal of Operational Research* 287, 1161–1169.
- 889 Viswanathan, G.M., Buldyrev, S.V., Havlin, S., Da Luz, M., Raposo, E.,
890 Stanley, H.E., 1999. Optimizing the success of random searches. *Nature*
891 401, 911–914.
- 892 Wang, L., Zeng, Y., Chen, T., 2015. Back propagation neural network with
893 adaptive differential evolution algorithm for time series forecasting. *Expert
894 Systems with Applications* 42, 855–863.
- 895 Zamuda, A., Sosa, J.D.H., 2019. Success history applied to expert system
896 for underwater glider path planning using differential evolution. *Expert
897 Systems with Applications* 119, 155–170.