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 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

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

Hence, the MDTWNPP objective function to be minimized is

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

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), (Kratica 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 $i \in \{1, \ldots, n\}$, 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 Scleanection 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 (Karmarker 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 probability 0.1, every bit of an offspring, which is further improved by a purposely defined heuristic operator; finally, the (μ, λ) replacement strategy is adopted. The GA outperformed CPLEX in the largest instances with $n \geq 400$.

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

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

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

The second meta-heuristic uses an Electromagnetism-like (EM) approach. A solution is represented as a real vector in $[0, 1]^n$, which is decoded to a binary partition by means of a simple thresholding procedure: the vector v_i of the MDTWNPP instance is assigned to the set S_0 when the i-th solution component is smaller than 0.5, otherwise it is assigned to S_1. At each generation, every individual undergoes to local search and scaling operators, then all the solutions are moved according to "electromagnetic forces" that can be attractive or repulsive depending on the objective values in the current population. It is worthwhile to note that the real vector encoding of EM is highly redundant because any MDTWNPP solution may be represented by an infinite number of real vectors.

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

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

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

To the best of our knowledge, the state-of-the-art MDTWNPP algorithm to date is MADEB, i.e., the memetic algebraic differential evolution proposed in our preliminary work (Santucci et al., 2019) of which this article is an extension. In fact, MADEB significantly outperformed GRASP+ePR, VNS and CPLEX on a set of 126 benchmark instances by obtaining the best average results on 106 instances (about 84% of the benchmark suite) and 76 new best known solutions. However, also MADEB adopts the simlpe redundant bit-string representation and the merely syntactic neighborhood definitions for its local search part.

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

2.2. Differential Evolution

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

DE evolves a population of N real vectors $\{x_1, \ldots, x_N\}$ by iteratively applying 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,

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

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

Notably, equation (3) perturbs x_{base} by an amount that is obtained from the differences' distribution of the DE population which is, itself, constantly evolved during the search. This mechanism allows DE to continuously adapt its mutation strength and it is the reason of why the differential mutation is 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

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

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

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

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

3. Main scheme of iMADEB

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

The focal point for the non-redundant representation is that a generic partition $\{S_0, S_1\}$ is equivalent to $\{S_1, S_0\}$. In fact, under the objective function definition provided in equation (2), $f(S_0, S_1) = f(S_1, S_0)$. Though this aspect is easy to read, all the previous population-based meta-heuristics 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 $S = \{v_1, v_2, \ldots, v_n\}$ 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:         $y_i \leftarrow$ AlgebraicDifferentialMutation($x_i$)
 5:         $z_i \leftarrow$ VariableNeighborhoodDescent($y_i$)
 6:     end for
 7:     for $i \leftarrow 1$ to $N$ do
 8:         $x_i \leftarrow$ 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 $x_{best}$
```

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

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

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

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

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

4. Binary Algebraic Differential Mutation

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

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

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

4.1. Previous algebraic operators for the binary space

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

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

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

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

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

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

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

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

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

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

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

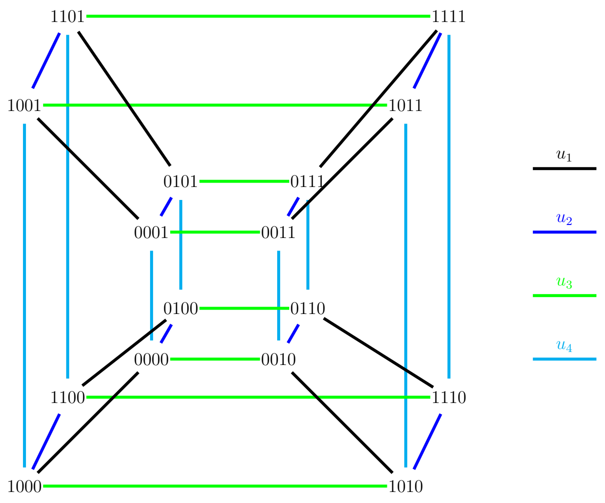


Figure 1: Cayley graph G_1 for the redundant repr. (i.e., using the generating set U)

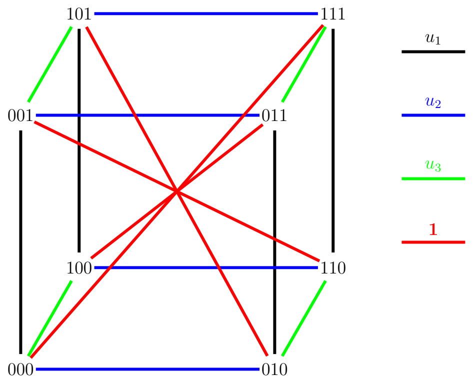


Figure 2: Cayley graph G_2 for the non-redundant repr. (i.e., using the generating set $\hat{U}$)

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 $(0101) \veebar u_2 = (0001)$.

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

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

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

4.3. Search characteristics of the binary differential mutation in iMADEB

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

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

Their genotypic difference is $\delta = x_{r_1} \ominus x_{r_2} = x_{r_1} \veebar x_{r_2} = (1111100)$ and, using Algorithm 2, is factorized as $\hat{U}_\delta = \{u_6, u_7, \mathbf{1}\}$. In fact, it is easy to see 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 of the MDTWNPP vectors corresponding to the generators in $\hat{U}_\delta$, i.e., the vectors v_6, v_7 and v_8. Therefore, the weight of the difference bit-string is $|\delta| = |\hat{U}_\delta| = 3$.

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

Finally, $y_i = x_i \oplus (F \odot \delta) = (0101010) \veebar (0000011) = (0101001)$. 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

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

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 the MDTWNPP vectors v_6 and v_7.

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

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

5. Lévy flight adaptation

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

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

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

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

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

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

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

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

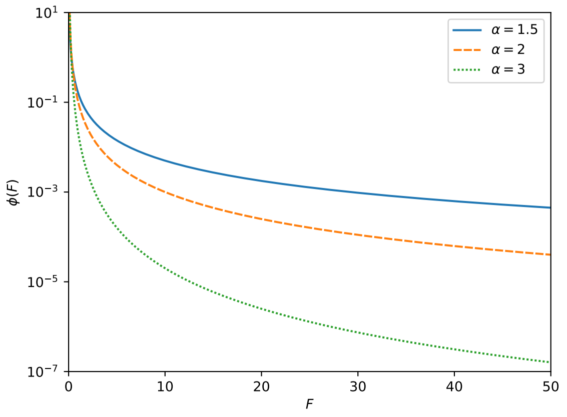


Figure 3: Power-law density (in log-scale) for $\alpha \in \{1.5, 2, 3\}$

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

6. Variable Neighborhood Descent

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

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

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

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

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

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

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

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

Algorithm 3 Pseudocode of VariableNeighborhoodDescent

```
1: function VariableNeighborhoodDescent($y \in \mathbb{B}^{n-1}$)
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
```

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

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

7. Experiments

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

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

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

7.1. Tuning and analysis of the iMADEB parameters

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

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

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

Every iMADEB setting has been executed 25 times per instance with a computational budget of 240 seconds per execution. Therefore, 24000 executions have been carried out for a total of 9600 hours of computational time.

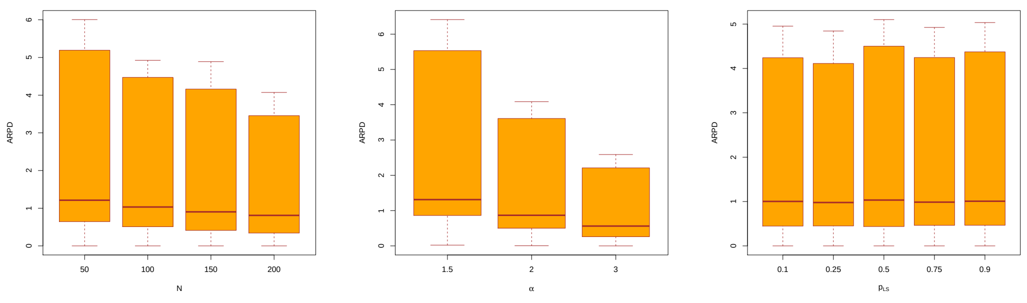
The performance of each iMADEB setting `S`, on every instance `i`, is measured by the commonly adopted average relative percentage deviation

(ARPD) index, defined as

$$\mathtt{ARPD}_{\mathtt{i}}^{\mathtt{S}} = \frac{1}{25} \sum_{j=1}^{25} \frac{\mathtt{S}_{\mathtt{i}}^{j} - \mathtt{Best}_{\mathtt{i}}}{\mathtt{Best}_{\mathtt{i}}} \times 100, \tag{8}$$

where $\mathtt{S}_{\mathtt{i}}^{j}$ is the objective value obtained by the iMADEB setting $\mathtt{S}$ in its j-th run on the instance $\mathtt{i}$, and $\mathtt{Best}_{\mathtt{i}}$ is the best objective value achieved among all the performed executions on instance $\mathtt{i}$.

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.



(a) Population size N (b) Lévy-flight parameter α (c) Local search prob. p_{LS}

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 N	α	p_{LS}	**Average Rank**	**Overall ARPD**	**Post-hoc 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

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

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

iMADEB has been executed using the setting of parameters identified in Section 7.1 and it is compared with the two state-of-the-art algorithms to date: MADEB (Santucci et al., 2019) and GRASP+ePR (Rodriguez et al., 2017). In order to perform a fair comparison, all the three algorithms have been executed on the same machine and using the same budget of computational time. Moreover, the executable code of GRASP+ePR has been got from the website provided by the authors (`https://sci2s.ugr.es/MDTWNP`), and both GRASP+ePR and MADEB have been run with the parameters settings suggested in, respectively, (Santucci et al., 2019) and (Rodriguez et al., 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.

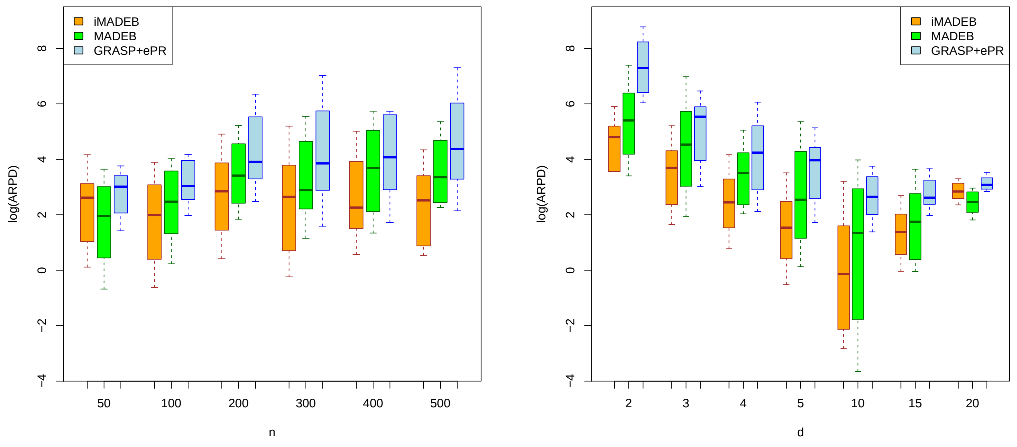
Table 2: Average ranks and number of best solutions obtained

	Average Rank			**No. Best Solutions**		
n	**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.

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



(a) log(ARPD) with instances grouped by n

(b) log(ARPD) with instances grouped by d

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

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

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

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

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

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

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

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

8. Conclusion and Future Work

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

Our proposal is motivated by a critical analysis of the MDTWNPP literature. In fact, all the previously proposed meta-heuristics adopt a redundant representation scheme for the solutions and do not consider the intrinsic characteristics of the MDTWNPP objective function in the design of the local search neighborhoods.

Table 3: Detailed experimental results on all the 210 instances

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	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	1249.66	**1198.17**	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	1024.80	**1000.26**	1015.52	1018.58	987.27
50_4e	**1193.75**	1204.30	1215.75	**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	2274.20	**2260.52**	2269.69	2269.69	2183.63
50_5e	**3959.34**	4404.84	4608.16	**3649.46** *	3655.57	4441.06	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	14125.54
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**	33211.07	30728.55
50_15b	39556.42	**36385.14**	36825.77	33240.10	34700.80	**33240.09** *	33240.09
50_15c	40996.84	**32470.97**	35748.85	29920.90	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**	35754.56	**25003.80** *	31800.70	31800.69	29018.28
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**	50382.38
50_20c	63775.68	**54141.94**	57999.14	50560.90	50560.90	**50560.86**	50560.86
50_20d	61134.74	**56093.25**	56785.88	53956.00	53956.00	**53955.96**	51538.57
50_20e	61339.02	**51239.31**	57688.77	**48281.50**	**48281.50**	**48281.50**	47829.86

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	GRASP +ePR	
100_2a	**0.62**	11.07	17.90	**0.44** *	7.68	15.35	4.60
100_2b	**0.77**	0.83	2.91	**0.56** *	0.56	0.85	0.57
100_2c	**0.62**	1.16	5.75	**0.26** *	0.26	2.28	0.26
100_2d	**0.00**	0.00	1.76	**0.00**	**0.00**	**0.00**	0.00
100_2e	**1.04**	1.07	1.52	**0.70**	1.06	1.07	0.40
100_3a	**105.45**	120.95	128.06	**84.00** *	115.00	124.15	115.00
100_3b	**76.90**	89.55	99.04	**60.20** *	82.68	88.78	88.78
100_3c	**166.81**	227.24	263.40	**58.94** *	216.26	234.56	68.11
100_3d	**228.34**	238.67	248.79	**216.53** *	234.83	231.78	234.83
100_3e	**115.47**	135.00	144.10	**94.58** *	129.20	138.36	132.25
100_4a	**493.91**	508.29	520.13	**467.26** *	502.18	511.35	508.28
100_4b	**868.16**	878.82	998.51	**845.10** *	872.55	887.81	875.60
100_4c	**872.34**	899.24	906.17	**752.64** *	892.98	902.13	892.98
100_4d	**1079.43**	1092.10	1099.27	**1056.55** *	1087.06	1093.17	1090.12
100_4e	**483.89**	510.27	518.79	**444.20** *	505.24	514.39	502.28
100_5a	**2410.86**	2426.38	2991.69	**2386.55**	2420.12	2429.28	2347.35
100_5b	**2064.91**	2075.95	2356.14	**2050.15** *	2068.48	2083.73	2077.63
100_5c	**2751.22**	2813.46	2832.25	**2742.26** *	2785.91	2817.96	2757.52
100_5d	**2941.45**	2968.92	2975.08	**2628.09** *	2966.78	2966.78	2637.39
100_5e	**2528.46**	3771.42	3778.00	**2422.43** *	3764.89	3770.99	2440.74
100_10a	**15990.02**	15995.22	16616.69	**15980.60**	15986.70	15986.68	15792.74
100_10b	**13589.79**	13595.95	17929.94	**13575.90** *	13588.20	13591.20	13591.20
100_10c	**11985.23**	14221.92	15387.30	**11968.80** *	11993.20	15377.41	11984.02
100_10d	**14962.60**	15550.88	16863.09	**14882.90** *	15543.90	15543.85	14898.14
100_10e	**12180.65**	15100.11	16132.32	**12169.90** *	12176.00	15822.36	12179.01
100_15a	**31994.82**	32151.06	33340.65	**31091.80**	31113.10	32152.39	30286.52
100_15b	29940.16	**28980.29**	31940.31	**28699.40**	28711.60	28711.59	27893.84
100_15c	**32038.63**	32885.44	34143.51	**29784.70**	31121.50	31115.38	29778.60
100_15d	32055.68	**31757.74**	34330.90	**30687.90** *	**30687.90** *	30690.91	30690.91
100_15e	31007.17	**30374.22**	32501.96	**28898.70**	30250.60	30256.44	28393.88
100_20a	54534.58	**52436.03**	55359.41	**49333.70**	49336.80	50231.79	43945.36
100_20b	55136.75	**53703.14**	56596.52	49833.10	48349.00	**45409.02** *	45673.84
100_20c	54213.54	**51745.85**	56007.33	48775.40	**46340.70**	46598.68	44839.37
100_20d	53899.90	**50172.85**	53883.93	47457.40	**43789.70**	47454.38	43786.66
100_20e	54445.41	**51616.68**	55913.30	**46151.00**	48095.10	49691.05	43923.29

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	GRASP +ePR	
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	0.00
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	**25.69** *	99.30	111.50	47.04
200_3e	**77.66**	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**	1904.98	1943.95	**1202.05** *	1897.51	1909.71	1217.32
200_5b	**1831.75**	1861.72	2624.69	**1751.33** *	1855.15	1876.53	1852.11
200_5c	**3.00**	3.13	6.29	**1.52** *	3.05	3.05	3.05
200_5d	**2069.86**	2119.54	2613.65	**1885.14** *	2106.26	2369.47	2112.36
200_5e	**1203.18**	2935.78	3784.08	**1185.27** *	2038.66	3777.25	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** *	17387.20	18098.25	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**	30474.63	30881.96	**30083.70** *	30465.30	30470.06	30464.39
200_15d	24055.90	**23563.88**	31658.01	**22822.70** *	22825.80	22831.88	22834.94
200_15e	29235.59	**29024.40**	32404.98	**28602.50**	**28602.50**	28608.64	27483.37
200_20a	47708.70	**46793.39**	51427.54	**40391.60**	42048.50	42066.76	40388.54
200_20b	48847.39	**48592.36**	51358.95	**41671.70** *	42099.30	46026.19	42311.89
200_20c	49039.63	**48137.58**	50790.92	**36926.60** *	41999.80	45571.04	36938.80
200_20d	**49267.76**	49608.93	50697.35	**39539.10** *	41186.90	41168.59	39539.13
200_20e	49550.85	**48439.67**	51614.55	41777.30	**37156.10** *	41777.28	39408.54

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	GRASP +ePR	
300_2a	**0.19**	0.20	1.28	**0.18** *	0.19	0.20	0.20
300_2b	**0.50**	0.58	2.14	**0.18** *	0.55	0.79	0.44
300_2c	**0.48**	0.68	2.98	**0.08** *	0.34	0.75	0.08
300_2d	**0.31**	0.58	2.82	**0.16** *	0.23	0.68	0.43
300_2e	**0.45**	0.69	2.22	**0.19** *	0.31	0.92	0.31
300_3a	**1.52**	1.68	5.62	**1.52** *	1.52	1.52	1.52
300_3b	**59.63**	77.70	93.38	**38.78** *	69.34	79.62	73.52
300_3c	**36.27**	113.53	254.82	**20.81** *	54.37	211.67	53.45
300_3d	**27.47**	97.94	122.42	**16.49** *	62.31	108.45	50.10
300_3e	**122.83**	165.07	242.67	**84.82** *	145.85	185.53	141.67
300_4a	**1.93**	2.82	7.02	**1.52** *	1.53	4.58	3.06
300_4b	**850.80**	968.63	1069.18	**821.10**	885.19	1058.92	563.08
300_4c	**556.60**	884.43	900.00	**387.60** *	876.20	882.29	430.31
300_4d	**613.39**	984.32	1058.87	**483.73** *	975.59	1007.45	673.67
300_4e	**642.87**	877.58	912.45	**620.15** *	863.09	893.60	717.82
300_5a	**1.98**	3.13	7.45	**1.52** *	1.53	4.57	3.04
300_5b	**2036.89**	2111.91	3199.13	**1987.76** *	2101.68	2711.21	2018.29
300_5c	**891.57**	901.96	917.12	**874.27** *	889.55	901.75	901.75
300_5d	**1873.22**	2099.09	2447.68	**907.43** *	2081.85	2351.16	1927.34
300_5e	**1512.06**	1531.80	1571.77	**1482.64** *	1516.23	1534.55	1528.44
300_10a	**14398.50**	14405.05	15364.03	**14382.50**	14394.70	14406.89	12839.36
300_10b	**14841.41**	14851.27	16251.03	**14823.80** *	14836.00	14860.42	14851.27
300_10c	**14093.90**	15966.58	15989.48	**13843.40**	15959.60	15977.88	13843.39
300_10d	**12871.28**	15821.42	18391.57	**10317.40** *	15809.70	17884.70	15812.71
300_10e	**15116.39**	15292.08	16813.31	**14811.80**	15279.60	15288.72	14805.66
300_15a	**27389.33**	27658.07	31796.17	**23878.30** *	27436.20	27442.27	27445.32
300_15b	**28871.27**	29077.31	31040.42	**28646.00** *	28854.30	29183.56	28863.48
300_15c	**22435.48**	26835.85	32004.47	**22208.50** *	22217.60	30725.11	22220.71
300_15d	**28358.48**	32226.38	34265.01	**27833.60** *	30415.00	32727.36	27873.22
300_15e	28775.63	**28771.88**	30621.06	**27640.40** *	28754.30	28763.50	28754.34
300_20a	45905.39	**45330.07**	48198.45	41141.60	**41135.50** *	42470.76	41405.18
300_20b	48599.51	**46665.25**	50529.23	**42587.40** *	42986.60	44133.93	44127.83
300_20c	45885.55	**44382.21**	49212.70	**34241.50** *	34253.70	43334.23	34247.61
300_20d	48320.13	**47172.10**	51162.27	**42351.40** *	44799.80	46825.40	42351.44
300_20e	46959.16	**46436.68**	50098.26	**40923.80**	43332.00	43350.29	37132.46

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	GRASP +ePR	
400_2a	**0.07**	0.13	7.01	**0.07** *	0.08	0.09	0.09
400_2b	**0.27**	0.48	1.57	**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	**440.83**	874.37	900.68	**383.02** *	865.52	886.88	437.95
400_4d	**548.54**	789.44	898.03	**264.88** *	779.46	872.84	794.72
400_4e	**629.24**	868.88	913.59	**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	**862.47** *	895.64	907.85	902.13
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**	1573.68	1595.03	1465.10
400_10a	**14117.06**	14547.91	15203.17	**13791.70**	14274.80	14809.11	7701.83
400_10b	**14812.86**	17043.77	18757.56	**14526.10**	17029.90	17908.98	10827.98
400_10c	**11671.89**	15589.55	17115.57	**10778.50**	14503.10	15672.45	9202.61
400_10d	**13712.60**	13722.33	15201.44	**13699.00**	13708.10	15176.29	8212.91
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	**26811.97**	27133.33	30252.59	**26779.00** *	26785.10	26821.74	26812.58
400_15d	**25904.86**	29986.69	33462.53	**22226.10**	29973.40	29997.82	16172.37
400_15e	**28712.23**	30700.24	30927.16	**28214.10** *	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**	44445.43	48047.22	**40322.20**	40325.30	43124.38	37938.93
400_20c	**45280.90**	46655.78	50454.56	**41868.70**	44834.60	44998.25	32762.87
400_20d	44787.21	**44002.46**	48454.07	**41297.60**	41387.50	41396.68	36726.27
400_20e	44206.29	**42089.62**	48320.45	**38942.30** *	39541.10	39535.05	39541.15

Instance	Average Obj. Values			Best Obj. Values			Previous Best Solution
	iMADEB	MADEB	GRASP +ePR	iMADEB	MADEB	GRASP +ePR	
500_2a	**0.12**	0.33	2.04	**0.11** *	0.12	1.03	0.12
500_2b	**0.28**	0.54	1.76	**0.11** *	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	**0.22**	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	1.53
500_4b	**814.17**	1144.45	1173.06	**752.87** *	1137.89	1156.23	845.27
500_4c	**421.21**	619.27	1106.60	**241.40** *	599.89	1051.03	606.00
500_4d	**1.93**	2.14	8.06	**1.52** *	1.52	3.04	1.53
500_4e	**469.17**	529.99	894.14	**390.64** *	509.69	611.97	534.12
500_5a	**2.44**	2.75	7.63	**1.52** *	1.52	3.06	3.06
500_5b	**1755.66**	1837.92	2358.82	**1315.21** *	1826.17	1868.89	1841.42
500_5c	**1720.18**	1870.90	2867.61	**1643.04** *	1854.88	2813.88	1879.29
500_5d	**2.69**	3.74	9.89	**1.52** *	1.53	7.62	3.05
500_5e	**866.96**	2499.49	2650.36	**801.92** *	2490.03	2541.92	817.16
500_10a	**12896.61**	12981.57	14177.87	**12874.20** *	12904.80	14159.93	12892.03
500_10b	**12107.12**	12112.50	12139.25	**12088.10**	12094.20	12115.56	10387.62
500_10c	**12801.53**	15239.90	17703.22	**12586.30**	14570.90	17679.18	10282.30
500_10d	**12800.82**	12865.19	13296.19	**12762.50** *	12845.40	12881.99	12823.52
500_10e	**14854.88**	15509.74	17445.43	**14154.20** *	15499.80	15545.61	14212.20
500_15a	**25988.84**	25994.29	30473.04	**25965.70**	25981.00	26032.89	20332.66
500_15b	**22739.25**	28653.36	31807.62	**22215.40** *	22258.10	29375.97	22258.13
500_15c	27116.44	**27108.86**	29400.01	**27094.40**	27097.40	27100.48	25461.22
500_15d	**26968.94**	29582.32	30645.53	**26324.00** *	29538.20	29722.39	26351.40
500_15e	**26496.11**	30150.42	30481.71	**24949.10**	30140.60	30131.50	21652.01
500_20a	43085.34	**42461.60**	48147.27	**38330.10**	38849.70	42063.29	32897.39
500_20b	43108.71	**41464.94**	48708.93	**35488.90** *	36651.00	44350.52	36626.58
500_20c	**42599.51**	42970.62	48087.84	**37584.30** *	37611.80	41591.09	37587.38
500_20d	**43664.83**	44643.25	48367.82	**34696.60** *	42189.00	43972.78	38774.75
500_20e	42412.02	**41547.43**	46541.12	**34812.70** *	39011.40	39026.66	38578.86

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

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

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

Experiments have been held in order to analyze iMADEB robustness and to compare its effectiveness with respect to the other state-of-the-art algorithms. Regarding robustness, though iMADEB has three parameters to be set, the experimental study carried out provides clear and robust indications for the practitioners that need to choose an iMADEB setting. Most importantly, the experimental comparison with the previously proposed approaches clearly show that iMADEB can be considered the new state-of-the-art algorithm for the MDTWNPP, both in terms of average and peak results. Moreover, comprehensive experimental data are also provided in order to facilitate comparisons.

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

References

Baioletti, M., Milani, A., Santucci, V., 2018. Learning bayesian networks with algebraic differential evolution, in: Proc. of 15th Int. Conf. on Parallel Problem Solving from Nature – PPSN XV, Springer International Publishing, Cham. pp. 436–448.

Baioletti, M., Milani, A., Santucci, V., 2020. Variable neighborhood algebraic differential evolution: An application to the linear ordering problem with cumulative costs. Information Sciences 507, 37–52.

Bi, Y., Srinivasan, D., Lu, X., Sun, Z., Zeng, W., 2014. Type-2 fuzzy multi-intersection traffic signal control with differential evolution optimization. Expert Systems with Applications 41, 7338–7349.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V., 2006. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10, 646–657.

Corus, D., Oliveto, P.S., Yazdani, D., 2018. Artificial immune systems can find arbitrarily good approximations for the np-hard partition problem, in: Proc. of 15th Int. Conf. on Parallel Problem Solving from Nature – PPSN XV, pp. 16–28.

Cuevas, E., Zaldivar, D., Pérez-Cisneros, M., 2010. A novel multi-threshold segmentation approach based on differential evolution optimization. Expert Systems with Applications 37, 5265–5271.

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

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

Hacibeyoglu, M., Tongur, V., Alaykiran, K., 2014. Solving the bi-dimensional two-way number partitioning problem with heuristic algorithms, in: 2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT), IEEE. pp. 1–5.

Hollander, M., Wolfe, D.A., Chicken, E., 2013. Nonparametric statistical methods. volume 751. John Wiley & Sons.

Iacca, G., dos Santos Junior, V.C., de Melo, V.V., 2020. An improved jaya optimization algorithm with lévy flight. Expert Systems with Applications 165, 113902.

Karmarker, N., Karp, R.M., 1983. The Differencing Method of Set Partitioning. Technical Report. USA.

Karp, R.M., 1972. Reducibility among combinatorial problems, in: Complexity of computer computations. Springer, pp. 85–103.

Kojić, J., 2010. Integer linear programming model for multidimensional two-way number partitioning problem. Computers & Mathematics with Applications 60, 2302–2308.

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

Mertens, S., 2006. The easiest hard problem: Number partitioning. Computational Complexity and Statistical Physics 125, 125–139.

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

Moscato, P., Cotta, C., 2003. A gentle introduction to memetic algorithms, in: Handbook of metaheuristics. Springer, pp. 105–144.

Moscato, P., Cotta, C., 2019. An accelerated introduction to memetic algorithms, in: Handbook of metaheuristics. Springer, pp. 275–309.

Moscato, P., Cotta, C., Mendes, A., 2004. Memetic algorithms, in: New optimization techniques in engineering. Springer, pp. 53–85.

Pavai, G., Geetha, T.V., 2016. A survey on crossover operators. ACM Computing Surveys 49, 1–43.

Penas, D., Banga, J., González, P., Doallo, R., 2015. Enhanced parallel differential evolution algorithm for problems in computational systems biology. Applied Soft Computing 33, 86–99.

Piotrowski, A.P., 2013. Adaptive memetic differential evolution with global and local neighborhood-based mutation operators. Information Sciences 241, 164–194.

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.

Pop, P.C., Matei, O., 2013b. A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem. Applied Mathematical Modelling 37, 9191–9202.

Price, K., Storn, R.M., Lampinen, J.A., 2006. Differential evolution: a practical approach to global optimization. Springer Science & Business Media.

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

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.

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.

Santucci, V., Baioletti, M., Milani, A., 2020. An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization. Swarm and Evolutionary Computation 55, 100673.

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.

Tanabe, R., Fukunaga, A., 2013. Success-history based parameter adaptation for differential evolution, in: Proc. of 2013 IEEE Congress on Evolutionary Computation – CEC 2013, IEEE. pp. 71–78.

Terpilowski, M.A., 2019. scikit-posthocs: Pairwise multiple comparison tests in python. Journal of Open Source Software 4, 1169.

Tomassini, M., 2016. Lévy flights in neutral fitness landscapes. Physica A: Statistical Mechanics and its Applications 448, 163–171.

Tsafarakis, S., Zervoudakis, K., Andronikidis, A., Altsitsiadis, E., 2020. Fuzzy self-tuning differential evolution for optimal product line design. European Journal of Operational Research 287, 1161–1169.

Viswanathan, G.M., Buldyrev, S.V., Havlin, S., Da Luz, M., Raposo, E., Stanley, H.E., 1999. Optimizing the success of random searches. Nature 401, 911–914.

Wang, L., Zeng, Y., Chen, T., 2015. Back propagation neural network with adaptive differential evolution algorithm for time series forecasting. Expert Systems with Applications 42, 855–863.

Zamuda, A., Sosa, J.D.H., 2019. Success history applied to expert system for underwater glider path planning using differential evolution. Expert Systems with Applications 119, 155–170.