Solving the Lexicographic Multi-Objective Mixed-Integer Linear Programming Problem Using Branch-and-Bound and Grossone Methodology

Marco Cococcioni[a], Alessandro Cudazzo[a], Massimo Pappalardo[a], Yaroslav D. Sergeyev[b,c,*]

[a] *University of Pisa, Pisa (Italy)*
[b] *University of Calabria, Rende (Italy)*
[c] *Lobachevsky State University of Nizhni Novgorod (Russia)*

Abstract

In the previous work (see [1]) the authors have shown how to solve a Lexicographic Multi-Objective Linear Programming (LMOLP) problem using the Grossone methodology described in [2]. That algorithm, called GrossSimplex, was a generalization of the well-known simplex algorithm, able to deal numerically with infinitesimal/infinite quantities.

The aim of this work is to provide an algorithm able to solve a similar problem, with the addition of the constraint that some of the decision variables have to be integer. We have called this problem LMOMILP (Lexicographic Multi-Objective Mixed-Integer Linear Programming).

This new problem is solved by introducing the GrossBB algorithm, which is a generalization of the Branch-and-Bound (BB) algorithm. The new method is able to deal with lower-bound and upper-bound estimates which involve infinite and infinitesimal numbers (namely, Grossone-based numbers). After providing theoretical conditions for its correctness, it is shown how the new method can be coupled with the GrossSimplex algorithm described in [1], to solve the original LMOMILP problem. To illustrate how the proposed algorithm finds the optimal solution, a series of LMOMILP benchmarks having a known solution is introduced. In particular, it is shown that the GrossBB combined with the GrossSimplex is able solve the proposed LMOMILP test problems with up to 200 objectives.

Keywords
Multi-Objective Optimization; Lexicographic Optimization; Mixed-Integer Linear Programming; Numerical Infinitesimals; Grossone Methodology

1. Introduction

It is well known that the Linear Programming, i.e., optimization of a linear function over a domain being the intersection of linear inequalities has attracted a lot of attention since World War II. At the end of '90, multi-objective optimization problems with conflicting objectives started to be under an intense investigation, especially using stochastic methods aiming at approximating the Pareto optimal frontier (see [3, 4, 5] and references given therein). Recently, lexicographic multi-objective optimization problems is gaining popularity (see [1, 6, 7, 8, 9]). The solution of the lexicographic multi-objective problem is also an optimal in the Pareto sense, but, of course, not all the Pareto optimal solutions are lexicographic optimal. Thus, being in general unique (when the problem is not multi-modal), the lexicographic optimum is particularly interesting to find. In a previous work [1], the lexicographic multi-objective linear programming problem (LMOLP) has been solved by introducing the GrossSimplex algorithm being a generalization of the well-known simplex algorithm able to deal with infinitesimal/infinite quantities modeled using Grossone methodology (see [2]). The main idea of that work was to transform the multi-objective problem into a single-objective one, where the objectives were summed up with infinitesimal weights, and the order of the infinitesimal weights decreased with the decrease of the importance of the objectives.

In the present work, we investigate the case where some of the decision variables are integer. We have called such class of problems LMOMILP (Lexicographic Multi-Objective Mixed-Integer Linear Programming). The integrality constraints hugely affect the problem, similarly to what happens in the

*Corresponding author. Tel.: +39 (0)984 494855.
Email addresses: marco.cococcioni@unipi.it (Marco Cococcioni), alessandro@cudazzo.com (Alessandro Cudazzo), massimo.pappalardo@unipi.it (Massimo Pappalardo), yaro@dimes.unical.it (Yaroslav D. Sergeyev)

single-objective case. Thus, we have decided to resort to the branch and bound (BB) approach, typically used to solve MILP (Mixed-Integer Linear Programming) problems. The key idea here is to call the GrossSimplex algorithm at each node visited by the BB algorithm to solve the relaxed problem (i.e., the one without integer constraints). However, the BB algorithm needs to be generalized in order to work with the GrossSimplex algorithm, since the latter returns as its output a bound for the optimal solution at the current node which can be a number not only with finite but also with infinitesimal components. The BB algorithm able to manage Grossone-based numbers and called GrossBB is introduced here, its pruning rules and the terminating conditions are described and studied. Finally, LMOMILP test problems having known solutions are proposed and it is shown on a number of numerical experiments that the GrossBB algorithm coupled with the GrossSimplex algorithm successfully find the correct solution. A preliminary version of the present work, significantly shorter and not containing the proof of the pruning rules contained herein, has been presented at the NUMTA'19 conference (see [10]).

In order to start, let us recall the basics of Grossone, the enabling methodology of this work. The numeral ① called *Grossone* has been introduced (see a recent survey [2]) as a basic element of a powerful numeral system allowing one to express not only finite but also different infinite and infinitesimal quantities (analogously, the numeral 1 is a basic element allowing one to express a variety of finite quantities). From the foundational point of view, grossone has been introduced as an infinite unit of measure equal to the number of elements of the set $\mathbb{N}$ of natural numbers (notice that the ①-based computational methodology is not related to non-standard analysis (see [11]) and its non-contradictory has been studied in depth in [12, 13, 14]). From the practical point of view, this methodology has given rise both to a new supercomputer patented in several countries (see [15]) and called *Infinity Computer* and to a variety of applications starting from optimization (see [1, 16, 17, 18, 19, 20, 21, 22]) and going through infinite series (see [2, 23, 24, 25, 26]), fractals and cellular automata (see [23, 27, 28, 29, 30, 31]), hyperbolic geometry and percolation (see [32, 33, 34]), the first Hilbert problem and Turing machines (see [2, 35, 36]), infinite decision making processes, game theory, and probability (see [37, 38, 39, 40, 41, 42]), numerical differentiation and ordinary differential equations (see [43, 44, 45, 46, 47]), etc.

The remaining text in the paper is structured as follows. In Section 2, the mixed-integer linear programming problem (MILP) is stated and the standard BB algorithm is presented briefly. In Section 3, the Lexicographic Multi-Objective Mixed-Integer Linear Programming (LMOMILP) problem is formalized. The Grossone methodology is briefly presented in Section 4, while Section 5 presents the GrossBB algorithm and its pruning rules, terminating conditions and branching rule. Section 6 presents five LMOMILP test problems and their solutions obtained using the proposed algorithm, whereas Section 7 is devoted to conclusions.

2. Mixed-Integer Linear Programming: MILP

An integer programming problem is a mathematical optimization problem in which some or all of the variables are restricted to be integers. In many settings where both the objective function and the constraints are linear the terminology is the following: integer linear programming (ILP), if all variables in the problem statement should be integers; mixed-integer linear programming (MILP) if only a subset of them should be integer.

2.1. The MILP problem

The MILP problem can be formalized as follows:

$$\begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} \\ \text{s. t.} & \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \\ & \mathbf{x} = \begin{bmatrix}\mathbf{p}\\ \mathbf{q}\end{bmatrix} \qquad \mathbf{p} \in \mathbb{Z}^k, \quad \mathbf{q} \in \mathbb{R}^{n-k}, \end{array} \qquad \mathcal{P}$$

where $\mathbf{c}$ is a column vectors$\in\mathbb{R}^n$, $\mathbf{x}$ is a column vector$\in\mathbb{R}^n$ (but k variables are constrained to be integer), $\mathbf{A}$ is a full-rank matrix$\in\mathbb{R}^{m\times n}$, $\mathbf{b}$ is a column vector$\in\mathbb{R}^m$. Hereinafter we assume that the feasibility region of problem $\mathcal{P}$ is bounded and non-empty. As in any MILP problem, from the problem $\mathcal{P}$, we can define the polyhedron with linear constraints:

$$\mathcal{S} \equiv \{\mathbf{x}\in\mathbb{R}^n : \mathbf{A}\mathbf{x} \leqslant \mathbf{b}\}. \tag{1}$$

Let us now introduce the new problem $\mathcal{R}$, which is a relaxed version of problem $\mathcal{P}$. Namely, it is obtained from $\mathcal{P}$ by removing the integrality constraint:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \end{aligned} \qquad \mathcal{R}$$

There are different techniques to find the optimal value of a MILP problem or an approximation, one of these is the BB algorithm, which, as explained in the next subsection, solves the relaxed problems $\mathcal{R}$ associated to a series of new sub-problems derived from $\mathcal{P}$.

2.2. MILP solved using the LP-based BB algorithm

Before introducing the LMOMILP problem and the GrossBB algorithm, let us recall the MILP problem and its solution based on the BB algorithm, combined with an LP solver. When a MILP problem $\mathcal{P}$ is bounded and non-empty (as we have assumed before), the total number of feasible solutions is finite. The BB approach is based on the principle that the total set of feasible solutions can be partitioned into smaller subsets of solutions. These smaller subsets can then be evaluated systematically until the best solution is found. The BB approach is coupled with a Linear Programming (LP) solver when is used to solve a MILP problem.

This method employs a tree structure (generally binary), nodes and branches are used as framework for the solution process. We define the problem $\mathcal{P}$ as the root problem and with P_{ij} the problem at node (ij), the nodes are enumerated and visited with the Breadth-First-Search (BFS) approach (P_{ij} refers to the j-th problem at level i of the tree). First of all, we will calculate the lower bound $v_I(\mathcal{P})$ by solving the relaxation of $\mathcal{P}$ and the upper bound $v_S(\mathcal{P})$ determined with a greedy algorithm (or we will assign to it the value $+\infty$).

The optimal solution $v(\mathcal{P})$ will thus always be between these two values (*integrality gap*):

$$v_I(\mathcal{P}) \leqslant v(\mathcal{P}) \leqslant v_S(\mathcal{P}). \tag{2}$$

Hereinafter, since we shall use always binary trees, we will indicate with P_c the current problem to solve (the one at leaf node (P_c)), and with P_l and P_r the corresponding sub-problems at its left and right, respectively.

The BB algorithm uses the following pruning rules, terminating conditions and branching rule:

Theorem 1 (Pruning rules)**.** *Let $\mathbf{x}_{opt}$ be the best solution found so far for $\mathcal{P}$ and let be $v_S(\mathcal{P}) = \mathbf{c}^T\mathbf{x}_{opt}$ be the current upper bound. Considering the current node (P_c) and the associated problem P_c:*

1. *If the feasible region of P_c is empty the sub-tree with root (P_c) has no feasible solutions with a value lower than $\mathbf{c}^T\mathbf{x}_{opt}$. So we can prune this node.*
2. *If $v_I(P_c) \geqslant v_S(\mathcal{P})$, then we can prune at node (P_c), since the sub-tree with root (P_c) cannot have feasible solutions having a value lower than $v_S(\mathcal{P})$.*
3. *If $v_I(P_c) < v_S(P)$ and the optimal solution $\bar{\mathbf{x}}$ of the relaxed problem R_c of problem P_c is feasible for $\mathcal{P}$, then $\bar{\mathbf{x}}$ is a better candidate solution for $\mathcal{P}$ and thus we can update $\mathbf{x}_{opt}$ ($\mathbf{x}_{opt} = \bar{\mathbf{x}}$) and the value of the upper bound ($v_S(\mathcal{P}) = v_I(P_c)$). Finally, prune this node according to the second rule.*

It can be proved that the three pruning rules above are correct (see [48]).

Terminating conditions for the BB

1. **All the remaining leaves have been visited**: if all the leaves have been visited the BB algorithm stops.
2. **Maximum number of iteration reached**: when a given maximum number of iterations (provided by the user at the beginning) has been reached, the BB stops.
3. $\varepsilon-$**optimality reached**: when the normalized difference between the global lower bound and the global upper bound is close enough to zero, we can stop:

$$\Delta(\mathcal{P}) = \frac{v_S(\mathcal{P}) - v_I(\mathcal{P})}{|v_S(\mathcal{P})|} \leqslant \epsilon, \tag{3}$$

where the global lower bound $v_I(\mathcal{P})$ can be computed at any step as the minimum of the lower bounds in the queue of the problems to be solved.

Branching rule for the BB
If $v_I(P_c) < v_S(\mathcal{P})$ and none of the pruning rules have been applied, take $\bar{\mathbf{x}}$, the optimal solution of the relaxation at that node and branch on the component having the highest fractional part among those variables having integer restrictions. In case of ties, branch on the first component. Thus we will create two distinct sub-problems of the current problem, denoted as P_l and P_r. The pseudo-code for the BB algorithm is provided in Algorithm 1.

Algorithm 1 The LP-based BB Algorithm

Inputs: maxIter and a specific MILP problem $|P|$, to be put within the root node $(\mathcal{P})$
Outputs: $\mathbf{x}_{opt}$ (the optimal solution), f_{opt} (the optimal value)

Step 0. Insert $|P|$ into a queue of the sub problems that must be solved. Put $v_S(\mathcal{P}) = \infty$, $\mathbf{x}_{opt} = [\ ]$, and $f_{opt} = \infty$ or use a greedy algorithm to get an initial feasible solution.

Step 1a. If all the remaining leaves have been visited (empty queue), or the maximum number of iterations has been reached, or the $\varepsilon-$optimality condition holds, then goto Step 4. Otherwise extract from the head of the queue the next problem to solve and call it P_c (*current problem*). Remark: this policy of insertion of new problems at the tail of the queue and the extraction from its head leads to a *breadth-first* visit for the binary tree of the generated problems.

Step 1b. Solve R_c, the relaxed version of the problem P_c at hand, using the GrossSimplex and get $\bar{\mathbf{x}}$ and f_c $(= \mathbf{c}^T\bar{\mathbf{x}})$:

$$[\bar{\mathbf{x}}, f_c, \texttt{emptyPolyhedron}] \leftarrow \texttt{LPSolver}(R_c)$$

Step 2a. If the LP solver has found that the polyhedron is empty, then prune the sub-tree of (P_c) (according to Pruning Rule 1) by going to Step 1a (without branching (P_c)). Otherwise, we have found a new lower value for P_c:

$$v_I(P_c) = f_c$$

Step 2b. If $v_I(P_c) \geqslant v_S(\mathcal{P})$, then prune the sub-tree under P_c (according to Pruning Rule 2), by going to Step 1a (without branching P_c).

Step 2c. If $v_I(P_c) < v_S(\mathcal{P})$ and all components of $\bar{\mathbf{x}}$ that must be integer are actually ϵ-integer (i.e., $\bar{\mathbf{x}}$ is feasible), then we have found a better upper bound estimate. Thus we can update the value of $v_S(\mathcal{P})$ as:

$$v_S(\mathcal{P}) = v_I(P_c).$$

In addition we set $\mathbf{x}_{opt} = \bar{\mathbf{x}}$ and $f_{opt} = v_I(P_c)$. Then we also prune the sub-tree under (P_c) (according to Pruning Rule 3) by going to Step 1a (without branching (P_c)).

Step 3. If $v_I(P_c) < v_S(\mathcal{P})$ but *not* all components of $\bar{\mathbf{x}}$ that must be integer are actually ϵ-integer, we have to branch. Select the component of $\bar{x}_t$ of $\bar{\mathbf{x}}$ having the greatest fractional part, among all the components that must be integer. Create two new nodes (i.e., problems) with a new constraint for this variable, one with a new $\leqslant$ constraint for the rounded down value of $\bar{x}_t$ and another with a new $\geqslant$ constraint for the rounded up value of $\bar{x}_t$. Let us call the two new problems P_l and P_r and put them at the tail of the queue of the problems to be solved, then goto Step 1a.

Step 4. End of the algorithm.

---

3. Lexicographic Multi-Objective Mixed-Integer Linear Programming: LMOMILP

In this section we introduce the LMOMILP problem, which is stated as follows:

$$P \quad \begin{array}{ll} \text{lexmin} & \mathbf{c}^{1\,T}\mathbf{x},\, \mathbf{c}^{2\,T}\mathbf{x}, ..., \mathbf{c}^{r\,T}\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \\ & \mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \qquad \mathbf{p} \in \mathbb{Z}^k, \quad \mathbf{q} \in \mathbb{R}^{n-k} \end{array}$$

where $\mathbf{c}^i,\ i = 1, ..., r$, are column vectors $\in \mathbb{R}^n$, $\mathbf{x}$ is a column vector $\in \mathbb{R}^n$, $\mathbf{A}$ is a full-rank matrix $\in \mathbb{R}^{m \times n}$, $\mathbf{b}$ is a column vector $\in \mathbb{R}^m$. lexmin in P denotes *Lexicographic Minimum* and means that the first objective is much more important than the second, which is, on its turn, much more important than the third one, and so on. Sometimes in literature this is denoted as $\mathbf{c}^{1\,T}\mathbf{x} \gg \mathbf{c}^{2\,T}\mathbf{x} \gg ... \gg \mathbf{c}^{r\,T}\mathbf{x}$.

As in any MILP problem, from the problem P, we can define the polyhedron defined by the linear constraints alone:

$$\mathcal{S} \equiv \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leqslant \mathbf{b}\}. \tag{4}$$

Thus we can define problem R, the *relaxation* of a lexicographic (mixed) integer linear problem, obtained from P by removing the integrality constraint on each variable:

$$\begin{array}{ll} \text{lexmin} & \mathbf{c}^{1\,T}\mathbf{x},\ \mathbf{c}^{2\,T}\mathbf{x}, ..., \mathbf{c}^{r\,T}\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \end{array} \qquad R$$

Problem R is called LMOLP (Lexicographic Multi-Objective Linear Problem), and can be solved, as shown in [1], using the Grossone-based methodology from [2]). In addition, every thing we have said above for the MILP problem is still valid for the LMOMILP.

Notice that the formulation of P makes no use of Gross-numbers or Gross-arrays involving ①, namely, it involves finite numbers only. Hereinafter we assume that $\mathcal{S}$ is bounded and non-empty. In the next section we briefly introduce the Grossone methodology, that will be used in Section 5 to transform problem P into an equivalent formulation, based on the use of Grossone, which will be solved by the GrossBB algorithm to be introduced in Section 5.

4. The Grossone-based Methodology

In [2, 49, 50, 51] a computational methodology working with an infinite unit of measure called Grossone and indicated by the numeral ① has been introduced as the number of elements of the set of natural numbers $\mathbb{N}$. On the one hand, this allows one to treat easily many problems related to the traditional set theory operating with Cantor's cardinals. In the new framework, instead of the usage of cardinals the number of elements of infinite sets using ①-based numerals can be computed. For instance, the following sets that the traditional cardinalities identify as countable can be measured more precisely (see [2, 35, 50]). In fact, it can be shown that the set of even numbers $\mathbb{E}$ has $\frac{①}{2}$ elements, namely, two times less than the set of natural numbers having ① elements. The set of integers $\mathbb{Z}$ has $2①+1$ elements, the set $\mathbb{G}$ of square natural numbers

$$\mathbb{G} = \{x : x = n^2, x \in \mathbb{N}, n \in \mathbb{N}\}$$

has $\lfloor\sqrt{①}\rfloor$ elements, etc. Analogously, it becomes possible to discern among sets having the traditional cardinality of continuum infinite sets with different number of elements. For instance, it follows that the set of numbers $x \in [0, 1)$ expressed in the binary positional numeral system is equal to $2^①$ and the set of numbers $x \in [0, 1)$ expressed in the decimal positional numeral system has $10^① > 2^①$ elements (for more examples see [2, 35, 50, 51]).

On the other hand, in the numeral system built upon Grossone, there is the opportunity to treat infinite and infinitesimal numbers in a unique framework and to work with all of them numerically, i.e., by executing arithmetic operations with floating-point numbers and the possibility to assign concrete infinite and infinitesimal values to variables. This is one of the differences with Robinson's Non-Standard Analysis where non-standard infinite numbers are discussed but, if K is a non-standard infinite integer, there is no possibility to assign a value to K, it always remains just a symbol without any concrete numerical value and only symbolic computations can be executed with it (see [11] for a detailed discussion).

The new numeral ① is introduced by describing its properties (following the same approach that lead to the introduction of zero in the past to switch from natural to integer numbers). To introduce Grossone, three methodological postulates and The Infinite Unit Axiom is added to the axioms of real numbers (see [2]). In particular, this axiom states that for any given finite integer n the infinite number $\frac{①}{n}$ is integer being larger than any finite number. Since the axiom is added to the standard axioms of real numbers, all standard properties (commutative, associative, existence of inverse, etc.) also apply to ① and Grossone-based numerals. Instead of the usual symbol ∞ different infinite and/or infinitesimal numerals can be used thanks to ①. Indeterminate forms are not present and, for example, the following

relations hold for infinite numbers ①, $①^2$ and infinitesimals $①^{-1}$, $①^{-2}$, as for any other (finite, infinite, or infinitesimal) number expressible in the new numeral system:

$$0 \cdot ① = ① \cdot 0 = 0, \quad ① - ① = 0, \quad \frac{①}{①} = 1, \quad ①^0 = 1, \quad 1^{①} = 1, \quad 0^{①} = 0,$$

$$0 \cdot ①^{-1} = ①^{-1} \cdot 0 = 0, \quad ①^{-1} > ①^{-2} > 0, \quad ①^{-1} - ①^{-1} = 0, \quad 2① - ① = ①,$$

$$\frac{①^{-1}}{①^{-1}} = 1, \quad (①^{-1})^0 = 1, \quad ① \cdot ①^{-1} = 1, \quad ① \cdot ①^{-2} = ①^{-1},$$

$$\frac{5①^{-2}}{①^{-2}} = 5, \quad \frac{60.1①^2}{①} = 60.1①, \quad \frac{①^{-1}}{2①^{-2}} = 0.5①, \quad ①^2 \cdot ①^{-1} = ①, \quad ①^2 \cdot ①^{-2} = 1.$$

A general way to express infinities and infinitesimals is also provided in [2, 49, 50, 51] by using records similar to traditional positional number systems, but with the radix ①. A number $\tilde{c}$ in this new numeral system ($\tilde{c}$ will be called Gross-scalar from here on) can be constructed by subdividing it into groups of corresponding powers of ① and thus can be represented as

$$\tilde{c} = c_{p_m}①^{p_m} + ... + c_{p_1}①^{p_1} + c_{p_0}①^{p_0} + c_{p_{-1}}①^{p_{-1}} + ... + c_{p_{-k}}①^{p_{-k}},$$

where $m, k \in \mathbb{N}$, exponents p_i are called Gross-powers (they can be numbers of the type of $\tilde{c}$) with $p_0 = 0$, and $i = m, ..., 1, 0, -1, ..., -k$. Then, $c_{p_i} \neq 0$ called Gross-digits are finite (positive or negative) numbers, $i = m, ..., 1, 0, -1, ..., -k$. In this numeral system, finite numbers are represented by numerals with the highest Gross-power equal to zero, e.g., $-6.2 = -6.2①^0$. Infinitesimals are represented by numerals having negative finite or infinite Gross-powers. The simplest infinitesimal is $①^{-1}$ for which $①^{-1} \cdot ① = 1$. We notice that all infinitesimals are not equal to zero, e.g., $①^{-1} > 0$. A number is infinite if it has at least one positive finite or infinite Gross-power. For instance, the number $43.6①^{4.56①} + 16.7①^{3.6} - 3.2①^{-2.1}$ is infinite, it consists of two infinite parts and one infinitesimal part.

In the context of this paper the following definition is important. A Gross-number (Gross-scalar) is said *purely finite* iff the coefficient associated with the zeroth power of Grossone is the only one to be different from zero. For instance, the number 3.4 is purely finite and $3.4 - 3.2①^{-2.1}$ is finite but not purely finite since it has an infinitesimal part.

5. LMOMILP solved using the GrossSimplex-based GrossBB algorithm

First of all, let us introduce the new problem $\tilde{P}$, formulated using Gross-numbers:

$$\begin{array}{ll} \min & \tilde{\mathbf{c}}^T\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \\ & \mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \qquad \mathbf{p} \in \mathbb{Z}^k, \quad \mathbf{q} \in \mathbb{R}^{n-k}, \end{array} \qquad \tilde{P}$$

where $\tilde{\mathbf{c}}$ is a column Gross-vector having n Gross-scalar components built using purely finite vectors $\mathbf{c}^i$

$$\tilde{\mathbf{c}} = \sum_{i=1}^{r} \mathbf{c}^i ①^{-i+1} \tag{5}$$

and $\tilde{\mathbf{c}}^{\mathbf{T}}\mathbf{x}$ is the Gross-scalar obtained by multiplying the Gross-vector $\tilde{\mathbf{c}}$ by the purely finite vector $\mathbf{x}$

$$\tilde{\mathbf{c}}^{\mathbf{T}}\mathbf{x} = (\mathbf{c}^{1T}\mathbf{x})①^0 + (\mathbf{c}^{2T}\mathbf{x})①^{-1} + ... + (\mathbf{c}^{rT}\mathbf{x})①^{-r+1}, \tag{6}$$

where (6) can be equivalently written in the extended form as:

$$\tilde{\mathbf{c}}^{\mathbf{T}}\mathbf{x} = (c_1^1 x_1 + ... + c_n^1 x_n)①^0 + (c_1^2 x_1 + ... + c_n^2 x_n)①^{-1} + ... + (c_1^r x_1 + ... + c_n^r x_n)①^{-r+1}.$$

What makes the new formulation $\tilde{P}$ attractive is the fact that its relaxed (from the integrality constraint) version is a Gross-LP problem (see [1]), which can be effectively solved using *a single run* of the GrossSimplex algorithm proposed in [1]. This means that the set of multiple objective functions is mapped into a single (Gross-) scalar function to be optimized. This opens the possibility to solve the

integer-constrained variant of the problem using an adaptation of the BB algorithm (see Alg. 2), coupled with the GrossSimplex. Of course, the GrossSimplex will solve the relaxed version of $\tilde{P}$:

$$\begin{aligned} \min \quad & \tilde{\mathbf{c}}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leqslant \mathbf{b} \end{aligned} \qquad \tilde{R}$$

The following Theorem 2 shows that problem $\tilde{P}$ is equivalent to problem P defined in Section 3.

Theorem 2 (Equivalence of problem $\tilde{P}$ and problem P)**.** *Problem $\tilde{P}$ is equivalent to the problem P and both of them have the same solution.*

Proof. The basic observation is that the integer relaxation R of problem P is an LMOLP problem, while the integer relaxation of problem $\tilde{P}$, the $\tilde{R}$ problem defined above, is a Gross-LP problem. In [1] we have already proved the equivalence of problems R and $\tilde{R}$. Since problems P and $\tilde{P}$ have equivalent relaxations, the two will also share the same solutions when the same integrality constraints will be taken into account on both. □

In the next subsections we will provide the pruning rules, terminating conditions and the branching rule. Then we will introduce the GrossBB algorithm being a generalization of the BB algorithm able to work with Gross-numbers.

5.1. Pruning rules for the GrossBB

The pruning rules presented above can be adapted to the GrossBB algorithm as follows.

Theorem 3 (Pruning rules for the GrossBB)**.** *Let $\mathbf{x}_{opt}$ be the best solution found so far for $\tilde{P}$, and let $\tilde{v}_S(\tilde{P}) = \tilde{\mathbf{c}}^T \mathbf{x}_{opt}$ be the current upper bound. Considering the current node $(\tilde{P}_c)$ and the associated problem $\tilde{P}_c$ the following assertions hold:*

1. *If the feasible region of problem $\tilde{P}_c$ is empty the sub-tree with root $(\tilde{P}_c)$ has no feasible solutions having values lower than $\tilde{\mathbf{c}}^T \mathbf{x}_{opt}$. So we can prune this node.*
2. *If $\tilde{v}_I(\tilde{P}_c) \geqslant \tilde{v}_S(\tilde{P})$, then we can prune at node $(\tilde{P}_c)$, since the sub-tree with root $(\tilde{P}_c)$ cannot have feasible solutions having a value lower than $\tilde{v}_S(\tilde{P})$.*
3. *If $\tilde{v}_I(\tilde{P}_c) < \tilde{v}_S(\tilde{P})$ and the optimal solution $\bar{\mathbf{x}}$ of the relaxed problem $\tilde{R}_c$ is feasible for $\tilde{P}$, then $\bar{\mathbf{x}}$ is a better candidate solution for $\tilde{P}$, and thus we can update $\mathbf{x}_{opt}$ $(\mathbf{x}_{opt} = \bar{\mathbf{x}})$ and the value of the upper bound $(\tilde{v}_S(\tilde{P}) = \tilde{v}_I(\tilde{P}_c))$. Finally, prune this node according to the second rule.*

Proof. Let us prove the correctness of the pruning rules introduced above.

[Pruning Rule 1]. If the feasible region of the current problem $\tilde{R}_c$ is empty, then the one of $\tilde{P}_c$ is empty, too, since the domain of $\tilde{P}_c$ has additional constrains (the integrality constraints). Furthermore, all the domains of the problems in the leaves of the sub-tree having root in $\tilde{P}_c$ will be empty, as well, since the domains of the leaves have all additional constrains with respect to $\tilde{R}_c$. This proves the correctness of the first pruning rule. Now let us prove the second pruning rule.

[Pruning Rule 2]. Let us consider a generic leaf P_{leaf} of the sub-tree having root $\tilde{P}_c$. Let us indicate with $\tilde{v}(\tilde{P}_c)$ the optimal value of the current problem $\tilde{P}_c$ (the one with the integer constraints). Then the values at the leaves below $\tilde{P}_c$ must be greater than or equal to $\tilde{v}(\tilde{P}_c)$:

$$\tilde{v}(\tilde{P}_{leaf}) \geqslant \tilde{v}(\tilde{P}_c) \quad \forall \text{ leaf in SubTree}(\tilde{P}_c).$$

In fact, the domain of $\tilde{P}_c$ includes the ones of all the $\tilde{P}_{leaf}$, being each problem $\tilde{P}_{leaf}$ obtained by enriching $\tilde{P}_c$ with additional constraints. On the other hand, it follows that

$$\tilde{v}(\tilde{P}_c) \geqslant \tilde{v}_I(\tilde{P}_c),$$

since $\tilde{v}_I(\tilde{P}_c)$ is obtained as the optimal solution of $\tilde{R}_c$, a problem having a domain which includes the one of $\tilde{P}_c$. Thus, the following chain of inequalities always holds:

$$\tilde{v}(\tilde{P}_{leaf}) \geqslant \tilde{v}(\tilde{P}_c) \geqslant \tilde{v}_I(\tilde{P}_c) \quad \forall \text{ leaf in SubTree}(\tilde{P}_c)$$

Now, if $\tilde{v}_I(\tilde{P}_c) \geqslant \tilde{v}_S(\tilde{P})$, we can add an element to the chain:

$$\tilde{v}(\tilde{P}_{leaf}) \geqslant \tilde{v}(\tilde{P}_c) \geqslant \tilde{v}_I(\tilde{P}_c) \geqslant \tilde{v}_S(\tilde{P}) \quad \forall \text{ leaf in SubTree}(\tilde{P}_c)$$

from which we can conclude that

$$\tilde{v}(\tilde{P}_{leaf}) \succcurlyeq \tilde{v}_S(\tilde{P}) \quad \forall \text{ leaf in SubTree}(\tilde{P}_c).$$

This means that all the leaves of the current node will contain solutions that are worse (or equivalent) than the current upper bound. Thus the sub-tree rooted in $\tilde{P}_c$ can to be pruned (i.e., not explicitly explored). This proves the correctness of the second pruning rule.

Before proving the pruning rule 3, let us observe how the pruning rule above prevents from solving multi-modal problems, in the sense that with such pruning rule we are only able to find a single optimum, not all the solutions that might have the same cost function. In other words, the proposed pruning rule does not allow one to solve multi-modal problems, because we are deciding not to explore the sub-tree at a given node that could contain solutions having the same current optimal objective function value. To solve multi-modal problems, that rule must be applied only when

$$\tilde{v}_I(\tilde{P}_c) > \tilde{v}_S(\tilde{P}). \tag{7}$$

[Pruning Rule 3]. If $\tilde{v}_I(\tilde{P}_c) < \tilde{v}_S(\tilde{P})$ and $\bar{\mathbf{x}}$ is feasible for $\tilde{P}$, (i.e., if all the components of $\bar{\mathbf{x}}$ that must be integer are actually ϵ-integer), we have found a better estimate for the upper bound of $\tilde{P}$, and thus we can update it:

$$\tilde{v}_S(\tilde{P}) = \tilde{v}_I(\tilde{P}_c).$$

As a result, now $\tilde{v}_I(\tilde{P}_c) = \tilde{v}_S(\tilde{P})$, and thus:

$$\tilde{v}(\tilde{P}_{leaf}) \succcurlyeq \tilde{v}(\tilde{P}_c) \succcurlyeq \tilde{v}_I(\tilde{P}_c) = \tilde{v}_S(\tilde{P}) \quad \forall\, leaf \text{ in SubTree}(\tilde{P}_c).$$

Then again we have that the sub-tree having root $\tilde{P}_c$ cannot contain better solutions than $\bar{\mathbf{x}}$:

$$\tilde{v}(\tilde{P}_{leaf}) \succcurlyeq \tilde{v}_S(\tilde{P}) \quad \forall \text{ leaf in SubTree}(\tilde{P}_c).$$

This proves the correctness of the third pruning rule. □

5.2. Terminating conditions for the GrossBB

Let us now discuss the terminating conditions for the GrossBB algorithm. The first two are exactly the same of the classical BB, while the third requires some attention.

The terminating conditions are:

1. **All the remaining leaves have been visited**: if all the leaves have been visited the GrossBB algorithm stops.
2. **Maximum number of iteration reached**: when a given maximum number of iterations (provided by the user at the beginning) has been reached, the GrossBB stops.
3. $\tilde{\epsilon}$**−optimality reached**: when the normalized difference between the global lower bound and the global upper bound at the i-th iteration is close enough to zero, the GrossBB stops:

$$\tilde{\Delta}^i(\tilde{P}) = \frac{\tilde{v}_S(\tilde{P}) - \tilde{v}_I(\tilde{P})}{|\tilde{v}_S(\tilde{P})|} \preceq \tilde{\epsilon}, \tag{8}$$

where $\preceq$ is the component-wise less than or equal to operator defined among two Gross-scalars and different from the usual operator $\leqslant$ defined for ①-based numbers. In particular, equation (8) requires that *all* the Gross-digits of $\tilde{\Delta}^i(\tilde{P})$ are less or equal to Gross-digits of $\tilde{\epsilon}$. Let us make more comments upon computations executed in (8). It first involves the difference between two Gross-scalars. This intermediate result must be divided by the absolute value of $\tilde{v}_S(\tilde{P})$. While computing the absolute value of a Gross-scalar is straightforward, division (as it happens also in the traditional floating point arithmetic) requires more efforts (see [2]). The result of the Gross-division is a Gross-scalar that must be compared with the Gross-scalar $\tilde{\epsilon}$, which has the form

$$\tilde{\epsilon} = \epsilon_0 + \epsilon_1①^{-1} + \epsilon_2①^{-2} + ... + \epsilon_{r-1}①^{-r+1}.$$

Obviously, it is possible to chose $\epsilon_0 = \epsilon_1 = \epsilon_2 ... = \epsilon$, to simplify the presentation.

In order to illustrate the situation, let us see an example below. Suppose that we have a problem with three objectives (r=3) and $\epsilon = 10^{-6}$ has been chosen. Given the following

$$\tilde{\Delta}^i(\tilde{P}) = 1.1 \cdot 10^{-7} + 5 \cdot 10^{-3}①^{-1} + 1.7 \cdot 10^{-8}①^{-2}$$

it follows that $\tilde{\Delta}^i(\tilde{P}) \leqslant \tilde{\epsilon}$ but

$$\tilde{\Delta}^i(\tilde{P}) \not\preceq \tilde{\epsilon}$$

because the first-order infinitesimal component of $\tilde{\Delta}^i(\tilde{P})$, namely $(5 \cdot 10^{-3})$, is not less or equal to ϵ. Thus in this case the GrossBB algorithm cannot terminate: it will continue, trying to make *all* the components less or equal to ϵ.

5.3. Branching rule for the GrossBB

When the sub-tree below $\tilde{P}_c$ cannot be pruned (because it could contain better solutions), its sub-tree must be explored. Thus we have to branch the current node into P_l and P_r and to add these two new nodes to the tail of the queue of the sub-problems to be analyzed and solved by the GrossSimplex.

Algorithm 2 provides a pseudo-code for the GrossBB algorithm. Thus, the GrossBB algorithm using internally the GrossSimplex algorithm and the rules (pruning, terminating, branching) provided above is able to solve a given $\tilde{P}$ LMOMILP problem.

5.4. Final considerations before testing the algorithm

Let us conclude this section by introducing the concept of "epsilon integrality" and by commenting upon the usage of division among Gross-numbers in the next two subsections.

5.4.1. Epsilon integrality

The concept of $\epsilon-$integrality used in pruning rule 3 can be formalized as follows. A vector $\mathbf{x}$ is $\epsilon-$integer when all its components are $\epsilon-$integer. Its generic component x_i is $\epsilon-$integer when

$$x_i - \lfloor x_i \rfloor < \varepsilon \qquad \text{or} \qquad \lceil x_i \rceil - x_i < \varepsilon.$$

5.4.2. Division among Gross-numbers could be avoided

Division among Gross-numbers used in equation (8) can be avoided, by multiplying the two sides of the inequality by $|\tilde{v}_S(\tilde{P})|$:

$$\tilde{v}_S(\tilde{P}) - \tilde{v}_I(\tilde{P}) \;\preceq\; \tilde{\epsilon} \cdot |\tilde{v}_S(\tilde{P})|$$

This multiplication allows us to create a division-free variant of the GrossBB algorithm. We thank one of the anonymous reviewers for pointing this out. However, we guess that the use of division is still interesting from the theoretical point of view, because it allowed us to clarify better the concept of a Gross-number being "near zero". Furthermore, the impact of division in equation (8) on the overall computing time of the algorithm is not critical, since the overall computing time is mainly affected by the time required to compute solutions of the relaxed LMOLP problems.

Algorithm 2 The GrossBB Algorithm using GrossSimplex method internally

---

Inputs: maxIter and a specific LMOMILP problem $|\tilde{P}|$, to be put within the root node $(\tilde{P})$
Outputs: $\mathbf{x}_{opt}$ (the optimal solution, a purely finite vector), $\tilde{f}_{opt}$ (the optimal value, a Gross-scalar)

Step 0. Insert $|\tilde{P}|$ into a queue of the sub problems that must be solved. Put $\tilde{v}_S(\tilde{P}) = ①$, $\mathbf{x}_{opt} = [\ ]$, and $\tilde{f}_{opt} = ①$ or use a greedy algorithm to get an initial feasible solution.

Step 1a. If all the remaining leaves have been visited (empty queue), or the maximum number of iterations has been reached, or the $\tilde{\epsilon}$-optimality condition holds, then goto Step 4. Otherwise extract from the head of the queue the next problem to solve and call it $\tilde{P}_c$ (*current problem*). Remark: this policy of insertion of new problems at the tail of the queue and the extraction from its head leads to a *breadth-first* visit for the binary tree of the generated problems.

Step 1b. Solve $\tilde{R}_c$, the relaxed version of the problem $\tilde{P}_c$ at hand, using the GrossSimplex and get $\bar{\mathbf{x}}$ and $\tilde{f}_c$ $(= \tilde{\mathbf{c}}^T\bar{\mathbf{x}})$:

$$[\bar{\mathbf{x}}, \tilde{f}_c, \texttt{emptyPolyhedron}] \leftarrow \texttt{GrossSimplex}(\tilde{R}_c)$$

Step 2a. If the LP solver has found that the polyhedron is empty, then prune the sub-tree of $(\tilde{P}_c)$ (according to Pruning Rule 1) by going to Step 1a (without branching $(\tilde{P}_c)$). Otherwise, we have found a new lower value for $\tilde{P}_c$:

$$\tilde{v}_I(\tilde{P}_c) = \tilde{f}_c$$

Step 2b. If $\tilde{v_I}(\tilde{P}_c) \geqslant \tilde{v_S}(\tilde{P})$, then prune the sub-tree under $\tilde{P}_c$ (according to Pruning Rule 2), by going to Step 1a (without branching $\tilde{P}_c$).

Step 2c. If $\tilde{v_I}(\tilde{P}_c) < \tilde{v_S}(\tilde{P})$ and all components of $\bar{\mathbf{x}}$ that must be integer are actually ϵ-integer (i.e., $\bar{\mathbf{x}}$ is feasible), then we have found a better upper bound estimate. Thus we can update the value of $\tilde{v_S}(\tilde{P})$ as:

$$\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c).$$

In addition, we set $\mathbf{x}_{opt} = \bar{\mathbf{x}}$ and $\tilde{f}_{opt} = \tilde{v_I}(\tilde{P}_c)$. Then we also prune the sub-tree under $(\tilde{P}_c)$ (according to Pruning Rule 3) by going to Step 1a (without branching $(\tilde{P}_c)$).

Step 3. If $\tilde{v_I}(\tilde{P}_c) < \tilde{v_S}(\tilde{P})$ but *not* all components of $\bar{\mathbf{x}}$ that must be integer are actually ϵ-integer, we have to branch. Select the component $\bar{x}_t$ of $\bar{\mathbf{x}}$ having the greatest fractional part, among all the components that must be integer. Create two new nodes (i.e., problems) with a new constraint for this variable, one with a new $\leqslant$ constraint for the rounded down value of $\bar{x}_t$ and another with a new $\geqslant$ constraint for the rounded up value of $\bar{x}_t$. Let us call the two new problems $\tilde{P}_l$ and $\tilde{P}_r$ and put them at the tail of the queue of the problems to be solved, then goto Step 1a.

Step 4. End of the algorithm.

---

6. Experimental results

In this section, we first introduce five LMOMILP test problems having known solution. Then we verify that the GrossBB combined with the GrossSimplex is able to successfully solve these problems.

6.1. Test problem 1: the "kite" in 2D

This problem is a variation of the 2D problem with 3 objectives described in [8]:

$$\begin{array}{ll} \text{lexmax} & 8x_1 + 12x_2,\ 14x_1 + 10x_2,\ x_1 + x_2 \\ \text{s.t.} & 2x_1 + 1x_2 \leqslant 120 \\ & 2x_1 + 3x_2 \leqslant 210 + 2.5 \\ & 4x_1 + 3x_2 \leqslant 270 \\ & x_1 + 2x_2 \geqslant 60 \\ & -200 \leqslant x_1, x_2 \leqslant +200, \qquad \mathbf{x} \in \mathbb{Z}^n \end{array} \qquad |T_1|$$

The polygon $\mathcal{S}$ associated to this problem is shown in Fig. 1 (left sub-figure). The integer points (feasible solutions) are shown as black spots whereas the domain of the relaxed problem (i.e., without the integer constraints) is shown in light grey.

It can be seen that the first objective vector $\mathbf{c}^1 = [8, 12]^T$ is orthogonal to segment $[\alpha, \beta]$ ($\alpha = (0, 70.83), \beta = (28.75, 51.67)$) shown in the same figure. All the nearest integer points parallel to this segment are optimal for the first objective (see the right sub-figure in Fig. 1). Since the solution is not unique, there is the chance to try to improve the second objective vector ($\mathbf{c}^2 = [14, 10]^T$).

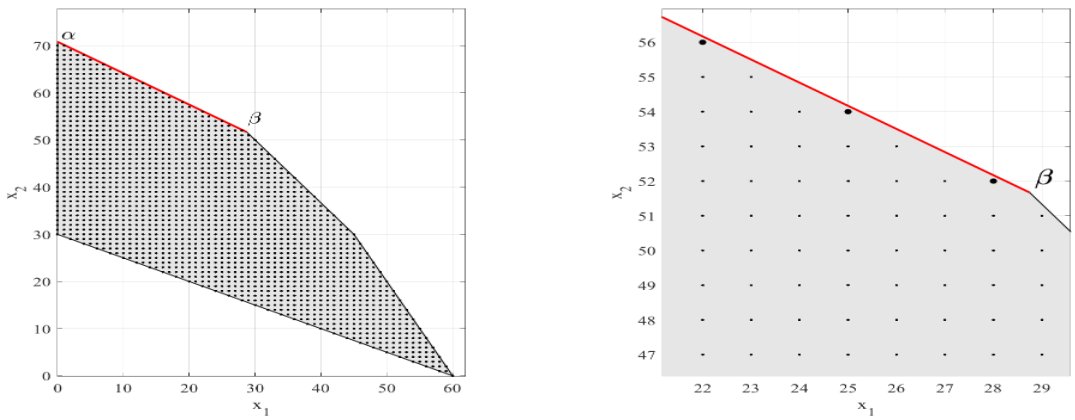

Figure 1: An example in two dimensions with three objectives. The black points on the left figure are all the feasible solutions. All the nearest integer points parallel to the segment $[\alpha, \beta]$ (there are many), are optimal for the first objective, while point (28, 52) is the unique lexicographic optimum for the given problem (i.e., if the second objective is considered, too). The third objective plays no role in this case. On the right, a zoom around point β is provided, with some optimal solutions for the first objective highlighted (the ones with a bigger black spot).

Let us see now what happens when we solve this problem using the GrossBB Algorithm with the GrossSimplex. notice, that since the $|T_1|$ problem is *lexmax*-formulated, we have to provide $-\tilde{\mathbf{c}}$ to the GrossBB algorithm.

Initialization. $\tilde{v_S}(\tilde{P}) = ①$, $\mathbf{x}_{opt} = [\ ]$, $\tilde{f}_{opt} = ①$ and insert $|T_1|$ into a queue of the sub-problems that must be solved.

Iteration 1. The GrossBB extracts from the queue of problems to be solved the only one present, and denotes it as the current problem: $\tilde{P}_c \equiv |T_1|$). Then the algorithm solves its relaxed version: the solution of $\tilde{R}_c$ is $\bar{\mathbf{x}} = [28.7500,\ 51.6667]^T$, with $\tilde{v_I}(\tilde{P}_c) = -850①^0 - 919.167①^{-1} - 80.4167①^{-2}$. As already seen on the MILP example, in this case we have to branch using the component having the highest fractional part (among the variables with integer restrictions, of course). In this case, it is the first component, and thus the new sub-problem on the left $\tilde{P}_l$ will have the additional constraint $x_1 \leqslant 28$, while the new on the right $\tilde{P}_r$ will have the additional constraint $x_1 \geqslant 29$. This split makes the current solution $[28.7500, 51.6667]^T$ not optimal neither for problems $\tilde{P}_l$ nor for $\tilde{P}_r$ (see Fig. 2).

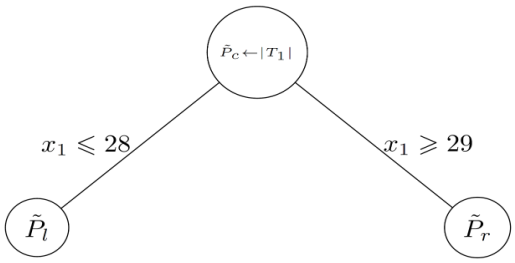

Figure 2: Situation at the end of iteration 1, for problem $|T_1|$.

Iteration 2. At this step the queue is composed by $[\tilde{P}_l, \tilde{P}_r]$, the problems generated in the previous iteration. The GrossBB extracts now the next problem from the top of the queue (*breadth-first* visit), namely $[\tilde{P}_l$ and denotes it as $\tilde{P}_c$, the current problem to solve. The optimal solution for the relaxed problem $\tilde{R}_c$ is $\bar{\mathbf{x}} = [28.25,\ 52]^T$, with $\tilde{v_I}(\tilde{P}_c) = -850①^0 - 915.5①^{-1} - 80.25①^{-2}$. We have to branch again, as we did during iteration 1, thus, a new left and right problems will be generated and added to the queue. The new left problem will have the additional constraint $x_1 \leqslant 28$, while the new right problem will have the additional constraint $x_1 \geqslant 29$. The length of the queue is now 3.

Iteration 3. Extract the next problem from the top of the queue and denote it as $\tilde{P}_c$. The optimal solution of $\tilde{R}_c$ is $\bar{\mathbf{x}} = [29.25,\ 51]^T$ and the associated $\tilde{v_I}(\tilde{P}_c) = -846①^0 - 919.5①^{-1} - 80.25①^{-2}$. We have to branch, new left and right problems will be generated, both will be added to the queue. The left problem will have the additional constraint $x_1 \leqslant 29$, while the new on the right $\tilde{P}_r$ will have the additional constraint $x_1 \geqslant 30$. The length of the queue is now 4.

Iteration 4. Extract the next problem from the queue and indicate it as $\tilde{P}_c$. Solve $\tilde{R}_c$, the relaxation of $\tilde{P}_c$, using the GrossSimplex. Since $\tilde{R}_c$ has an empty feasible region, prune this node by applying the first pruning rule. The length of the queue is now 3.

Iteration 5. Extract the next problem from the top of the queue and denote it as $\tilde{P}_c$. The optimal solution of $\tilde{R}_c$ is $\bar{\mathbf{x}} = [28,\ 52.1667]^T$ and the associated $\tilde{v_I}(\tilde{P}_c) = -850①^0 - 913.6667①^{-1} - 80.1667①^{-2}$. We have to branch: new left and right problems will be generated and added to the queue. The left problem will have the additional constraint $x_1 \leqslant 52$, while the right one will have the additional constraint $x_1 \geqslant 53$. The length of the queue is now 4.

Iteration 6. Extract the next problem from the queue and indicate it as $\tilde{P}_c$. This time the GrossSimplex returns an integer solution, i.e., a feasible solution for the LMOMILP initial problem $|T_1|$:

$$\bar{\mathbf{x}} = [30,\ 50]^T \ \text{ and } \ \tilde{v_I}(\tilde{P}_c) = -840①^0 - 920①^{-1} - 80①^{-2}$$

Since $\tilde{v_I}(\tilde{P}_c) < \tilde{v_S}(\tilde{P})$, then we can update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, $\mathbf{x}_{opt} = \bar{\mathbf{x}}$. Finally we can prune this node, according to the third pruning rule.

Iteration 7. Extract the next problem from the queue and denote it as $\tilde{P}_c$. Again the GrossSimplex returns an integer solution:

$$\bar{\mathbf{x}} = [29,\ 51]^T \ \text{ and } \ \tilde{v_I}(\tilde{P}_c) = -844①^0 - 916①^{-1} - 80①^{-2}$$

Since $\tilde{v_I}(\tilde{P}_c) < \tilde{v_S}(\tilde{P})$, then we can update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, $\mathbf{x}_{opt} = \bar{\mathbf{x}}$. Finally we can prune this node, according to the third pruning rule.

Iteration 8. Extract the next problem from the top of the queue and indicate it as $\tilde{P}_c$. The optimal solution of $\tilde{R}_c$ is $\bar{\mathbf{x}} = [26.75,\ 53]^T$, with $\tilde{v_I}(\tilde{P}_c) = -850①^0 - 904.5①^{-1} - 79.75①^{-2}$. We have to branch:

new left and right problems will be generated and added to the queue. The left problem will have the additional constraint $x_1 \leqslant 26$, while the new on the right $\tilde{P}_r$ will have the additional constraint $x_1 \geqslant 27$. The length of the queue is now 4.

Iteration 9. Extract the next problem from the queue ($\tilde{P}_c$) and denote it as the current problem $\tilde{P}_c$. Solve its relaxation using the GrossSimplex. In this case, the returned solution is feasible for the initial LMOMILP problem $|T_1|$ because it has all integral components:

$$\bar{\mathbf{x}} = [28,\, 52]^T \quad \text{and} \quad \tilde{v_I}(\tilde{P}_c) = -848①^0 - 912①^{-1} - 80①^{-2}.$$

Since $\tilde{v}_I(\tilde{P}_c) < \tilde{v}_S(\tilde{P})$, then update both $\tilde{v}_S(\tilde{P}) = \tilde{v}_I(\tilde{P}_c)$ and $\mathbf{x}_{opt} = \bar{\mathbf{x}}$. Finally, prune this node by applying the third pruning rule.

Iteration 10. Extract the next problem from the queue and indicate it as $\tilde{P}_c$. Solve $\tilde{R}_c$, the relaxation of $\tilde{P}_c$, using the GrossSimplex. Since $\tilde{R}_c$ has an empty feasible region, prune this node by applying the first pruning rule.

Iterations 11-79. The GrossBB algorithm is not able to find a better solution than the $\bar{\mathbf{x}} = [28,\, 52]^T$ already found, but continues to branch and explore the tree, until only two nodes remain in the queue. The processing of the last two nodes is discussed in the last two iterations 80 and 81, below.

Iteration 80. Extract the next problem from the queue and indicate it as $\tilde{P}_c$. Solve $\tilde{R}_c$ using the GrossSimplex. Since $\tilde{R}_c$ has an empty feasible region, prune this node by applying the first pruning rule.

Iteration 81. At this point there is one last unsolved problem from the queue. Extract this problem and denote it as $\tilde{P}_c$. The optimal solution of $\tilde{R}_c$ is:

$$\bar{\mathbf{x}} = [1,\, 70]^T, \quad \text{with} \quad \tilde{v_I}(\tilde{P}_c) = -848①^0 - 714①^{-1} - 71①^{-2}.$$

Since $\tilde{v_I}(\tilde{P}_c) \geq \tilde{v}_S(\tilde{P})$, prune this last node according to the third pruning rule. Being now the tree empty, the GrossBB algorithm stops according to the first terminating condition and returns the optimal solution found so far: $\mathbf{x}_{opt} = [28,\, 52]^T$. The optimal value of the objective function is $\tilde{\mathbf{c}}^{\mathbf{T}}\mathbf{x}_{opt} = 848①^0 + 912①^{-1} + 80①^{-2}$.

Table 1 provides a synthesis of the iterations performed by the GrossBB algorithm and described in detail above.

6.2. Test problem 2: the unrotated "house" in 3D

This illustrative example is in three dimensions with three objectives:

$$\begin{array}{lll} \text{lexmax} & x_1, -x_2, -x_3 & \\ \text{s.t.} & -10.2 \leqslant x_1 \leqslant 10.2 & \\ & -10 \leqslant x_2 \leqslant 10.2 & \\ & -10.2 \leqslant x_3 \leqslant 10.2 & |T_2| \\ & -x_1 - x_2 \leqslant 2 & \\ & -x_1 + x_2 \leqslant 2 & \\ & -20 \leqslant x_i \leqslant 20,\ i = 1, ..., 3, \qquad \mathbf{x} \in \mathbb{Z}^3 & \end{array}$$

with the domain being the cube shown in Fig. 3. It can be immediately seen that by considering the first objective alone (maximize x_1), all the nearest integer points parallel to square having vertices α, β, γ, δ are optimal for the first objective function (see Fig. 3). Since the optimum is not unique, the second objective function can be considered in order to improve it without deteriorating the first objective. Then, all the integer points close to the segment $[\beta,\, \gamma]$ are all optimal for the second objective, too (see Fig. 4, which provides the plant-view of Fig. 3 with $x_3 = -10$). Again, the optimum in not unique, and, therefore, the third objective is considered. This allows us to select the nearest integer point to γ as the unique solution that maximizes all the three objectives. The point $[10, -10, -10]$ is the lexicographic optimum to this problem.

The problem can be solved with GrossBB algorithm, as shown in Tab. 2. The solution $\mathbf{x}_{opt} = [10,\, -10,\, -10]^T$ is actually found after 5 iterations. The optimal value of the objective function is computed in the form $\tilde{\mathbf{c}}^T\mathbf{x}_{opt} = 10①^0 + 10①^{-1} + 10①^{-2}$.

6.3. Test problem 3: the rotated "house" in 5D

Algorithm 3 shows how to add a small rotation along the axis perpendicular to the plane containing the first two variables x_1 and x_2, for the "house" problem seen in previous example, after generalizing it to the n-dimensional case with $n = 5$.

Table 1: Iterations performed by the GrossBB Algorithm using GrossSimplex during solving problem $|T_1|$

Iteration	result at node(iteration)
Initialize	- $\tilde{v_S}(\tilde{P}) = ①$ - Queue len. 1 (add the root problem to the queue)
1	$\tilde{v_I}(\tilde{P}_c)$: $-850①^0 - 919.167①^{-1} - 80.4167①^{-2}$. Queue length : 0 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 2 - $\tilde{\Delta} = 100①^0 + 100①^{-1} + 100①^{-2}$
2	$\tilde{v_I}(\tilde{P}_c)$: $-850①^0 - 915.5①^{-1} - 80.25①^{-2}$. Queue length: 1 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 3 - $\tilde{\Delta} = 100①^0 + 100①^{-1} + 100①^{-2}$
3	$\tilde{v_I}(\tilde{P}_c)$: $-846①^0 - 919.5①^{-1} - 80.25①^{-2}$. Queue length: 2 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 4 - $\tilde{\Delta} = 100①^0 + 100①^{-1} + 100①^{-2}$
4	prune node: rule 1, empty feasible region. Queue length: 3
5	$\tilde{v_I}(\tilde{P}_c)$: $-850①^0 - 913.667①^{-1} - 80.1667①^{-2}$. Queue length: 2 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 4 - $\tilde{\Delta} = 100①^0 + 100①^{-1} + 100①^{-2}$
6	$\tilde{v_I}(\tilde{P}_c)$: $-840①^0 - 920①^{-1} - 80①^{-2}$. Queue length: 3 - A feasible solution has been found: $\mathbf{x}_{opt} = [30,\ 50]^T$ - update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, prune node: rule 3 - $\tilde{\Delta} = 0.01190048①^0 - 0.00688406①^{-1} + 0.00208333①^{-2}$
7	$\tilde{v_I}(\tilde{P}_c)$: $-844①^0 - 916①^{-1} - 80①^{-2}$. Queue length: 2 - A feasible solution has been found: $\mathbf{x}_{opt} = [29,\ 51]^T$ - update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, prune node: rule 3 - $\tilde{\Delta} = 0.354191①^0 - 0.127528①^{-1} + 0.104058①^{-2}$
8	$\tilde{v_I}(\tilde{P}_c)$: $-850①^0 - 904.5①^{-1} - 79.75①^{-2}$. Queue length: 1 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 3 - $\tilde{\Delta} = 0.007109①^0 - 0.00254731①^{-1} + 0.00208333①^{-2}$
9	$\tilde{v_I}(\tilde{P}_c)$: $-848①^0 - 912①^{-1} - 80①^{-2}$. Queue length: 2 - A feasible solution has been found: $\mathbf{x}_{opt} = [28,\ 52]^T$ - update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, prune node: rule 3 - $\tilde{\Delta} = 0.00235849①^0 - 0.00822368①^{-1} - 0.003125①^{-2}$
10	prune node: rule 1, empty feasible region. Queue length: 1
...	... ...
80	prune node: rule 1, empty feasible region. Queue length: 1
81	$\tilde{v_I}(\tilde{P}_c)$: $-848①^0 - 714①^{-1} - 71①^{-2}$. Queue length: 0 - $\tilde{v_I}(\tilde{P}_c) \geqslant \tilde{v_S}(\tilde{P})$ prune node: rule 2
result	Iteration 81. Optimization ended. Optimal solution found: $\mathbf{x}_{opt} = [28,\ 52]^T$ $\tilde{f}_{opt} = -848①^0 - 912①^{-1} - 80①^{-2}$ $\tilde{\Delta} = 0①^0 + 0①^{-1} + 0①^{-2}$

The problem consists of the lexicographic optimization of x_1, $-x_2$, ..., $-x_5$. The method used to generate a randomly rotated benchmark is shown in Alg. 3 (the generated rotation matrix $\mathbf{Q}$ is reported in Appendix A). After the rotation, the following lower and upper bounds were added

$$-2\rho \leqslant x_i \leqslant 2\rho,\ i = 1, ..., 5.$$

As a result, the following problem ($\mathbf{A}'$,$\mathbf{b}'$) has been generated:

$$\begin{aligned} \text{lexmax} \quad & x_1, -x_2, ..., -x_5 \\ \text{s.t.} \quad & \left\{\mathbf{x}' \in \mathbb{Z}^5 : \mathbf{A}'\mathbf{x}' \leqslant \mathbf{b}'\right\} \end{aligned} \qquad |T_3|$$

where $\mathbf{C}'$, $\mathbf{A}'$ and vector $\mathbf{b}'$ are reported in Appendix A as well.

The lexicographic optimum for this problem is $\mathbf{x}_{opt} = [1000, -999, -1000, -1000, -1000]^T$. The problem can be solved with GrossBB algorithm. After 11 steps, the algorithm has found the correct lexicographic optimum (see Tab. 3 in Appendix B).

Algorithm 3 Generation of a randomly rotated "house" problem in $\mathbb{R}^n$

Step 1. Let $\{\mathbf{Ax} \leqslant \mathbf{b},\ \mathbf{x} \in \mathbb{Z}^n\}$ be the initial, unrotated problem, in n-dimensions. The problems is formulated as follow (ρ is a parameter that controls the size of the house):

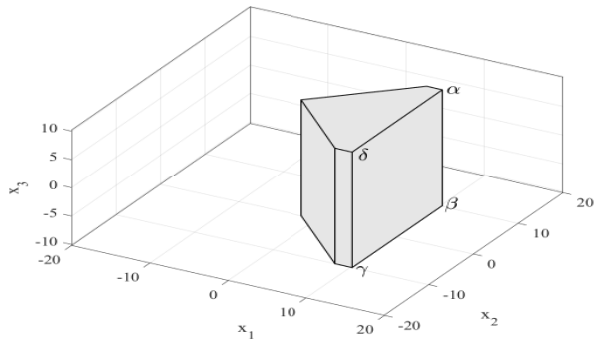

Figure 3: The 3D unrotated "house" problem.

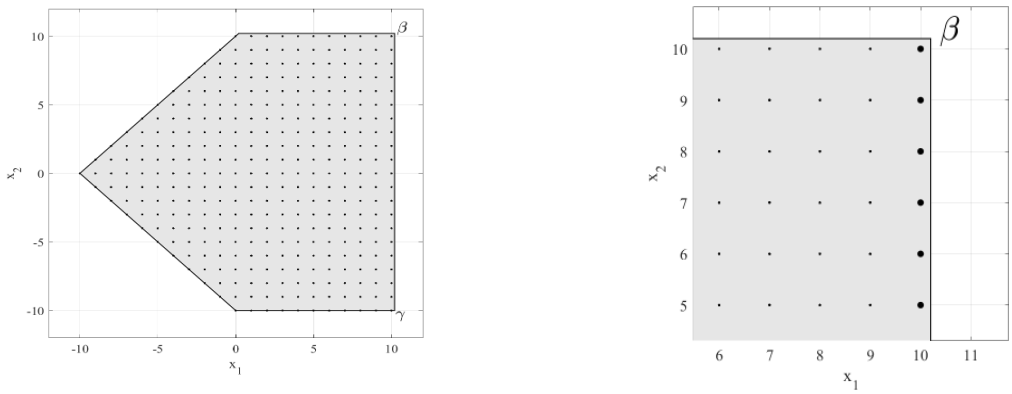

Figure 4: Section view of Fig. 3 with $x_3 = -10$ (left) and its top-right zoom (right).

$$
\begin{array}{ll}
\text{lexmax} & x_1,\, -x_2,\, ...,\, -x_n \\
\text{s.t.} & -\rho + 0.2 \leqslant x_1 \leqslant \rho + 0.2, \\
& -\rho \qquad\quad \leqslant x_2 \leqslant \rho + 0.2 \\
& -\rho + 0.2 \leqslant x_i \leqslant \rho + 0.2,\ i = 3, ..., n \\
& -x_1 - x_2 \leqslant 2 \\
& -x_1 + x_2 \leqslant 2 \\
& \mathbf{x} \in \mathbb{Z}^n
\end{array}
$$

Step 2. Use as rotation matrix $\mathbf{Q}$ with a random little rotation:

```
rA = 0.0002;

rB = 0.0005;

φ = (rB-rA).*rand(1) + rA;
```

$$
\mathbf{Q} = \begin{bmatrix}
\cos(\phi) & \sin(\phi) & 0 & 0 & .. & 0 \\
-\sin(\phi) & \cos(\phi) & 0 & 0 & ... & 0 \\
0 & 0 & 1 & 0 & ... & 0 \\
0 & 0 & 0 & 1 & ... & 0 \\
... & ... & ... & ... & ... & ... \\
0 & 0 & 0 & 0 & ... & 1
\end{bmatrix} \in \mathbb{R}^{n \times n}
$$

Step 3. Rotate the polytope: $\mathbf{A}' = \mathbf{AQ}$ ($\mathbf{b}$ and $\mathbf{C}$ does not change under rotations: $\mathbf{b}' = \mathbf{b}$ and $\mathbf{C}' = \mathbf{C}$) and then add these additional constraints as lower and upper bound for every variables to $\mathbf{A}'$ (they are twice the size of the house, in order to fully contain it):

$$-2\rho \leqslant x_i \leqslant 2\rho,\ i = 1, ..., n$$

Table 2: Iterations performed by GrossBB algorithm on test problem $|T_2|$

Iteration	result at node(iteration)
Initialize	- $\tilde{v_S}(\tilde{P}) = ①$ - Queue len. 1 (add the root problem to the queue)
1	$\tilde{v_I}(\tilde{P}_c)$: $-10.2①^0 - 10①^{-1} - 10.2①^{-2}$. Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 2 - $\tilde{\Delta}$ $100①^0 + 100①^{-1} + 100①^{-2}$
2	prune node: rule 1, empty feasible region. Queue length: 1
3	$\tilde{v_I}(\tilde{P}_c)$: $-10①^0 - 10①^{-1} - 10.2①^{-2}$. Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 2 - $\tilde{\Delta}$ $100①^0 + 100①^{-1} + 100①^{-2}$
4	$\tilde{v_I}(\tilde{P}_c)$: $-10①^0 - 10①^{-1} - 10①^{-2}$. Queue length: 1 - A feasible solution has been found: $x_{opt} = [10, -10, -10]^T$ - update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, prune node: rule 3 - $\tilde{\Delta}$: $0①^0 + 0①^{-1} + 0.02①^{-2}$
5	prune node: rule 1, empty feasible region. Queue length: 0
result	Iteration 5. Optimization ended. Optimal solution found: $\mathbf{x}_{opt} = [10, -10, -10]^T$ $\tilde{f}_{opt} = -10①^0 - 10①^{-1} - 10①^{-2}$ $\tilde{\Delta} = 0①^0 + 0①^{-1} + 0①^{-2}$

Step 4 For the unrotated problem the optimal integer solution is $\mathbf{x}_{opt} = [\rho, -\rho, -\rho, -\rho, ..., -\rho]^T$. When a rotation is applied, if the rotation is between a sufficiently small range of angles the optimal solution is: $\mathbf{x}_{opt} = [\rho, 1-\rho, -\rho, -\rho, ..., -\rho]^T$.
The optimal value is computed as: $\tilde{f}_{opt} = \tilde{\mathbf{c}}^T \mathbf{x}_{opt}$, where $\tilde{\mathbf{c}}$ is derived from $\mathbf{C}$.

6.4. Test problem 4: the randomly rotated hypercube in 7D

Algorithm 4 describes how to generate a test problem in $\mathbb{R}^n$, based on a randomly rotated hypercube having side 2000 and centered in the origin. The feasible region is further constrained to be within an unrotated hypercube, with lower side (200.4), centered in the origin, too. The randomly rotated external hypercube does not play any role, since the feasible region is governed by the inner hypercube, but adds complexity to the problem (i.e., it challenges more the GrossBB algorithm).

As an example, let us consider a hypercube in seven dimension. The corresponding rotation matrix $\mathbf{Q}$, matrix $\mathbf{A}'$, and vector $\mathbf{b}'$ are reported in Appendix A. The resulting problem ($|T_4|$) can be written as follows

$$\begin{aligned} &\text{lexmax } \mathbf{c'}^1 \cdot \mathbf{x}',\ \mathbf{c'}^2 \cdot \mathbf{x}',\ ...,\ \mathbf{c'}^7 \cdot \mathbf{x}', \\ &\text{s.t. } \left\{\mathbf{x}' \in \mathbb{Z}^7 : \mathbf{A}'\mathbf{x}' \leqslant \mathbf{b}'\right\}, \end{aligned} \qquad |T_4|$$

where $\mathbf{c'}^{1\,T}$ is the first row of $\mathbf{C}'$ reported in Appendix A, $\mathbf{c'}^{2\,T}$ is its second row, and so on. The lexicographic optimum for this problem is

$$\mathbf{x_{opt}} = [100, 100, 100, 100, 100, 100, 100]^T.$$

The GrossBB algorithm has been applied and the optimum has been obtained after 15 iterations, as shown in Tab. 4 (see Appendix C).

Algorithm 4 Generation of a randomly rotated hypercube in $\mathbb{R}^n$

Step 1. Let $\{\mathbf{Ax} \leqslant \mathbf{b},\ \mathbf{x} \in \mathbb{Z}^n\}$ be the initial, unrotated hypercube problem, in n-dimensions. The problem is formulated as follow:

$$\begin{aligned} &\text{lexmax} && x_1, x_2, ..., x_n \\ &\text{s.t.} && -1000 \leqslant x_i \leqslant 1000,\ i = 1, ..., n \\ & && \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

Step 2. Generate a random rotation matrix $\mathbf{Q}$. It can be computed using a QR factorization utility, applied to a random matrix $\mathbf{T}$. In particular, $\mathbf{T}$ must be an n-by-n matrix having entries randomly generated according to the normal distribution (zero mean and unitary variance). In Matlab the

matrix $\mathbf{Q}$ can be obtained in this way:

```
T = randn(n);
[Q, R] = qr(T);
```

Step 3. Rotate the polytope: $\mathbf{A}' = \mathbf{AQ}$ ($\mathbf{b}$ does not change under rotations: $\mathbf{b}' = \mathbf{b}$ and $\mathbf{C}' = \mathbf{C}$) and then add the inner hypercube by adding these constraints to $\mathbf{A}'$:

$$-(100 + 0.2) \leqslant x_i \leqslant (100 + 0.2),\ i = 1, ..., n.$$

Step 4. Compute the LMOMILP optimum:
$\mathbf{x}_{opt} = [100, 100, \ldots, 100]^T$
$\tilde{f}_{opt} = \tilde{\mathbf{c}}^T \mathbf{x}_{opt}$, where $\tilde{\mathbf{c}}$ is derived from $\mathbf{C}$.

6.5. Test problem 5: the randomly rotated hypercube in 200D

As a stress test for the GrossBB algorithm, we have applied it on a rotated hypercube problem with 200 objectives in $\mathbb{R}^{200}$ generated using Algorithm 4. The behaviour of the GrossBB is exactly the same as in the previous test problem in $\mathbb{R}^7$, but this time the solution is found after 401 iterations ($= 1 + 200 \cdot 2$) instead of after 15 ($= 1 + 7 \cdot 2$), where the number one is due to the fact that the problem is always solved at the root. Each time a problem is solved, two new sub-problems are generated by branching (the left and right ones). Due to this fact the number of iterations is twice the number of dimensions (other than the first iteration, of course). Indeed, for this particular problem, the left problems will always have an empty solution (once relaxed), while the right ones will always have feasible solutions (when relaxed), but these solutions will not be epsilon-integer. By construction, however, the feasible solution of the 200st right problem *will be* epsilon-integer, i.e., will be feasible for the original integer-constrained problem, and thus the algorithm will stop (this solution will be the vector $\mathbf{x}_{opt} \in \mathbb{R}^{200}$ equal to $[100, 100, ..., 100]^T$). For these reasons, the iterations of the GrossBB for this problem look very similar to those reported on Table 4 in Appendix C and thus we decided not to report them here due to space limitations. In this case, the GrossBB needed 32 hours and 13 min to complete the 401 iterations on an Intel i7 920 with 4 cores (8 threads) at 3.6 Ghz.

7. A brief conclusion

In the previous work [1], the lexicographic multiple objective linear programming problem (LMOLP) has been considered. To solve it, the GrossSimplex algorithm being a generalization of the simplex algorithm able to work with infinitesimals and infinities using the Grossone methodology has been proposed. In the present paper, the Lexicographic Multi-Objective Mixed-Integer Linear Programming Problem (called LMOMILP) has been considered. To solve it, the GrossBB algorithm has been introduced. This method is a generalization of the branch and bound to the case where not only finite but also infinitesimal and infinite numbers expressible in the numeral system using ① can be treated. It has been proved that the proposed pruning rules and the terminating conditions ensure the correct functioning of the GrossBB algorithm. Finally, five LMOMILP test problems having known solutions have been proposed and it was shown that the introduced GrossBB algorithm solves all of them successfully.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments.

Appendix A - Additional information for third and fourth test problems

This appendix reports information related to test problems 3 and 4. In order to construct the 5-dimensional problem $|T_3|$ the following rotation matrix $\mathbf{Q}$, matrices $\mathbf{A}'$, $\mathbf{C}'$, and vector $\mathbf{b}'$ have been used:

$$\mathbf{A}' = \begin{bmatrix}
0.9999999087 & 0.0004273220262 & 0 & 0 & 0 \\
-0.0004273220262 & 0.9999999087 & 0 & 0 & 0 \\
0 & 0 & 1.0 & 0 & 0 \\
0 & 0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 0 & 1.0 \\
-0.9999999087 & -0.0004273220262 & 0 & 0 & 0 \\
0.0004273220262 & -0.9999999087 & 0 & 0 & 0 \\
0 & 0 & -1.0 & 0 & 0 \\
0 & 0 & 0 & -1.0 & 0 \\
0 & 0 & 0 & 0 & -1.0 \\
-0.9995725867 & -1.000427231 & 0 & 0 & 0 \\
-1.000427231 & 0.9995725867 & 0 & 0 & 0 \\
-1.0 & 0 & 0 & 0 & 0 \\
0 & -1.0 & 0 & 0 & 0 \\
0 & 0 & -1.0 & 0 & 0 \\
0 & 0 & 0 & -1.0 & 0 \\
0 & 0 & 0 & 0 & -1.0 \\
1.0 & 0 & 0 & 0 & 0 \\
0 & 1.0 & 0 & 0 & 0 \\
0 & 0 & 1.0 & 0 & 0 \\
0 & 0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 0 & 1.0
\end{bmatrix} \quad \mathbf{b}' = \begin{bmatrix}
1000.2 \\ 1000.2 \\ 1000.2 \\ 1000.2 \\ 1000.2 \\ 1000.2 \\ 1000.0 \\ 1000.2 \\ 1000.2 \\ 1000.2 \\ 4.0 \\ 4.0 \\ 2000.0 \\ 2000.0 \\ 2000.0 \\ 2000.0 \\ 2000.0 \\ 2000.0 \\ 2000.0 \\ 2000.0 \\ 2000.0
\end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix}
0.9999999087 & 0.0004273220262 & 0 & 0 & 0 \\
-0.0004273220262 & 0.9999999087 & 0 & 0 & 0 \\
0 & 0 & 1.0 & 0 & 0 \\
0 & 0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 0 & 1.0
\end{bmatrix}$$

$$\mathbf{C}' = \begin{bmatrix} \mathbf{c}^1{}^T \\ \mathbf{c}^2{}^T \\ \dots \\ \mathbf{c}^5{}^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & .. & 0 \\ 0 & -1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & -1 \end{bmatrix} \in \mathbb{R}^{5\times 5}$$

The following matrices and vector are related to test problem $|T_4|$ in 7 dimensions:

$$\mathbf{A}' = \begin{bmatrix}
-0.3399 & -0.1993 & -0.1513 & -0.1745 & 0.1464 & 0.1863 & 0.8575 \\
0.1419 & -0.7158 & -0.1318 & 0.6014 & 0.2768 & -0.1028 & -0.03592 \\
-0.1374 & 0.3492 & -0.2176 & 0.6205 & -0.5398 & -0.2576 & 0.2627 \\
0.3681 & 0.08162 & -0.1831 & -0.26 & 0.2269 & -0.8112 & 0.2172 \\
-0.4154 & -0.1865 & 0.7852 & 0.02297 & -0.09166 & -0.407 & 0.03933 \\
0.5363 & 0.3372 & 0.4952 & 0.3105 & 0.2907 & 0.2384 & 0.3401 \\
0.4998 & -0.4133 & 0.1307 & -0.2414 & -0.6828 & 0.08847 & 0.1733 \\
0.3399 & 0.1993 & 0.1513 & 0.1745 & -0.1464 & -0.1863 & -0.8575 \\
-0.1419 & 0.7158 & 0.1318 & -0.6014 & -0.2768 & 0.1028 & 0.03592 \\
0.1374 & -0.3492 & 0.2176 & -0.6205 & 0.5398 & 0.2576 & -0.2627 \\
-0.3681 & -0.08162 & 0.1831 & 0.26 & -0.2269 & 0.8112 & -0.2172 \\
0.4154 & 0.1865 & -0.7852 & -0.02297 & 0.09166 & 0.407 & -0.03933 \\
-0.5363 & -0.3372 & -0.4952 & -0.3105 & -0.2907 & -0.2384 & -0.3401 \\
-0.4998 & 0.4133 & -0.1307 & 0.2414 & 0.6828 & -0.08847 & -0.1733 \\
-1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1.0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1.0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1.0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1.0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1.0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1.0 \\
1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1.0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1.0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1.0
\end{bmatrix} \quad \mathbf{b}' = \begin{bmatrix}
1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 1000.0 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2 \\ 100.2
\end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix}
-0.34 & -0.199 & -0.151 & -0.175 & 0.146 & 0.186 & 0.857 \\
0.142 & -0.716 & -0.132 & 0.601 & 0.277 & -0.103 & -0.0359 \\
-0.137 & 0.349 & -0.218 & 0.621 & -0.54 & -0.258 & 0.263 \\
0.368 & 0.0816 & -0.183 & -0.26 & 0.227 & -0.811 & 0.217 \\
-0.415 & -0.187 & 0.785 & 0.023 & -0.0917 & -0.407 & 0.0393 \\
0.536 & 0.337 & 0.495 & 0.31 & 0.291 & 0.238 & 0.34 \\
0.5 & -0.413 & 0.131 & -0.241 & -0.683 & 0.0885 & 0.173
\end{bmatrix}$$

$$\mathbf{C}' = \begin{bmatrix} \mathbf{c}^1{}^T \\ \mathbf{c}^2{}^T \\ \dots \\ \mathbf{c}^7{}^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & .. & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix} \in \mathbb{R}^{7\times 7}$$

Appendix B - Table 3 (GrossBB iterations on test problem $|T_3|$)

Iter.	result at node(iteration)
Initialize	- $\tilde{v_S}(\tilde{P}) = ①$ - Queue len. 1 (add the root problem to the queue)
1	$\tilde{v_I}(\tilde{P}_c)$: $-1000.63①^0 - 999.573①^{-1} - 1000.2①^{-2} - 1000.2①^{-3} - 1000.2①^{-4}$. Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4}$
2	$\tilde{v_I}(\tilde{P}_c)$: $-1000.63①^0 - 999①^{-1} - 1000.2①^{-2} - 1000.2①^{-3} - 1000.2①^{-4}$. Queue length: 1 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 3 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4}$
3	prune node: rule 1, empty feasible region. Queue length: 2
4	prune node: rule 1, empty feasible region. Queue length: 1
5	$\tilde{v_I}(\tilde{P}_c)$: $-1000①^0 - 999①^{-1} - 1000.2①^{-2} - 1000.2①^{-3} - 1000.2①^{-4}$. Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4}$
6	$\tilde{v_I}(\tilde{P}_c)$: $-1000①^0 - 999①^{-1} - 1000①^{-2} - 1000.2①^{-3} - 1000.2①^{-4}$. Queue length: 1 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 3 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4}$
7	prune node: rule 1, empty feasible region. Queue length: 2
8	$\tilde{v_I}(\tilde{P}_c)$: $-1000①^0 - 999①^{-1} - 1000①^{-2} - 1000①^{-3} - 1000.2①^{-4}$. Queue length: 1 - no pruning rules applied, branch $\tilde{P}_c$ in two sub-problems. Queue length: 3 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4}$
9	prune node: rule 1, empty feasible region. Queue length: 2
10	$\tilde{v_I}(\tilde{P}_c)$: $-1000①^0 - 999①^{-1} - 1000①^{-2} - 1000①^{-3} - 1000①^{-4}$. Queue length: 1 - A feasible solution has been found: $\tilde{\mathbf{x}}_{opt} = [1000\ -999\ -1000\ -1000\ -1000]^T$ - update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, prune node: rule 3 - $\tilde{\Delta} = 0①^0 + 0①^{-1} + 0①^{-2} + 0①^{-3} + 0.0002①^{-4}$
11	prune node: rule 1, empty feasible region. Queue length: 2
result	Iteration 11. Optimization ended. Optimal solution found: $\mathbf{x}_{opt} = [1000, -999, -1000, -1000, -1000]^T$ $\tilde{f}_{opt} = -1000①^0 - 999①^{-1} - 1000①^{-2} - 1000①^{-3} - 1000①^{-4}$ $\tilde{\Delta} = 0①^0 + 0①^{-1} + 0①^{-2} + 0①^{-3} + 0①^{-4}$

Appendix C - Table 4 (GrossBB iterations on test problem $|T_4|$)

Iter.	result at node(iteration)
Initialize	- $\tilde{v_S}(\tilde{P}) = ①$ - Queue len. 1 (add the root problem to the queue)
1	$\tilde{v_I}(\tilde{P}_c)$: $-100.2①^0 - 100.2①^{-1} - 100.2①^{-2} - 100.2①^{-3} - 100.2①^{-4} - 100.2①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
2	prune node: rule 1, empty feasible region Queue length: 1
3	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100.2①^{-1} - 100.2①^{-2} - 100.2①^{-3} - 100.2①^{-4} - 100.2①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
4	prune node: rule 1, empty feasible region Queue length: 1
5	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100①^{-1} - 100.2①^{-2} - 100.2①^{-3} - 100.2①^{-4} - 100.2①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
6	prune node: rule 1, empty feasible region Queue length: 1
7	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100①^{-1} - 100①^{-2} - 100.2①^{-3} - 100.2①^{-4} - 100.2①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
8	prune node: rule 1, empty feasible region Queue length: 1
9	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100①^{-1} - 100①^{-2} - 100①^{-3} - 100.2①^{-4} - 100.2①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
10	prune node: rule 1, empty feasible region Queue length: 1
11	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100①^{-1} - 100①^{-2} - 100①^{-3} - 100①^{-4} - 100.2①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
12	prune node: rule 1, empty feasible region Queue length: 1
13	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100①^{-1} - 100①^{-2} - 100①^{-3} - 100①^{-4} - 100①^{-5} - 100.2①^{-6}$ - Queue length: 0 - no pruning rules applied, branch $\tilde{P}_c$ in two subproblem: Queue length: 2 - Delta $100①^0 + 100①^{-1} + 100①^{-2} + 100①^{-3} + 100①^{-4} + 100①^{-5} + 100①^{-6}$
14	prune node: rule 1, empty feasible region Queue length: 1
15	$\tilde{v_I}(\tilde{P}_c)$: $-100①^0 - 100①^{-1} - 100①^{-2} - 100①^{-3} - 100①^{-4} - 100①^{-5} - 100①^{-6}$ - Queue length: 0 - A feasible solution has been found: $\mathbf{x}_{opt} = [100,\ 100,\ 100,\ 100,\ 100,\ 100,\ 100]^T$ - update $\tilde{v_S}(\tilde{P}) = \tilde{v_I}(\tilde{P}_c)$, prune node: rule 3 - Delta $0①^0 + 0①^{-1} + 0①^{-2} + 0①^{-3} + 0①^{-4} + 0①^{-5} + 0①^{-6}$
result	Iteration 15. Optimization ended. Optimal solution found: $\mathbf{x}_{opt} = [100,\ 100,\ 100,\ 100,\ 100,\ 100,\ 100]^T$ $\tilde{f}_{opt} = -100①^0 - 100①^{-1} - 100①^{-2} - 100①^{-3} - 100①^{-4} - 100①^{-5} - 100①^{-6}$ $\tilde{\Delta} = 0①^0 + 0①^{-1} + 0①^{-2} + 0①^{-3} + 0①^{-4} + 0①^{-5} + 0①^{-6}$

[1] M. Cococcioni, M. Pappalardo, and Y. D. Sergeyev, "Lexicographic multi-objective linear programming using grossone methodology: Theory and algorithm," *Applied Mathematics and Computation*, vol. 318, pp. 298–311, 2018.

[2] Y. D. Sergeyev, "Numerical infinities and infinitesimals: Methodology, applications, and repercussions on two Hilbert problems," *EMS Surveys in Mathematical Sciences*, vol. 4, pp. 219–320, 2017.

[3] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons Inc, 2 ed., 2001.

[4] M. Cococcioni, P. Ducange, B. Lazzerini, and F. Marcelloni, "A new multi-objective evolutionary algorithm based on convex hull for binary classifier optimization," in *Proc. 2007 IEEE Congress on Evolutionary Computation (IEEE-CEC'07)*, pp. 3150–3156, 2007.

[5] P. M. Pardalos, A. Žilinskas, and J. Žilinskas, *Non-Convex Multi-Objective Optimization*. Springer International Publishing, 2017.

[6] S. Khosravani, M. Jalali, A. Khajepour, A. Kasaiezadeh, S. K. Chen, and B. Litkouhi, "Application of lexicographic optimization method to integrated vehicle control systems," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 12, pp. 9677–9686, 2018.

[7] E. Weber, A. Rizzoli, R. Soncini-Sessa, and A. Castelletti, "A lexicographic optimization in water resource planning: the case of lake verbano, italy," in *Proc. 1st Biennial Meeting of the International Environmental Modelling and Software Society (IEMSS)*, 2002.

[8] I. Stanimirovic, "Compendious lexicographic method for multi-objective optimization," *Facta universitatis - series: Mathematics and Informatics*, vol. 27, no. 1, pp. 55–66, 2012.

[9] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3, pp. 317–343, 2011.

[10] M. Cococcioni, A. Cudazzo, M. Pappalardo, and Y. D. Sergeyev, "Grossone methodology for lexicographic mixed-integer linear programming problems," in *In Proc. of the the 3rd International Conference and Summer School on Numerical Computations: Theory and Algorithms*, June 2019.

[11] Y. D. Sergeyev, "Independence of the grossone-based infinity methodology from non-standard analysis and comments upon logical fallacies in some texts asserting the opposite," *Foundations of Science*, vol. 24, no. 1, pp. 153–170, 2019.

[12] G. Lolli, "Metamathematical investigations on the theory of grossone," *Applied Mathematics and Computation*, vol. 255, pp. 3–14, 2015.

[13] M. Margenstern, "Using grossone to count the number of elements of infinite sets and the connection with bijections," *p-Adic Numbers, Ultrametric Analysis and Applications*, vol. 3, no. 3, pp. 196–204, 2011.

[14] F. Montagna, G. Simi, and A. Sorbi, "Taking the Pirahã seriously," *Communications in Nonlinear Science and Numerical Simulation*, vol. 21, no. 1–3, pp. 52–69, 2015.

[15] Y. D. Sergeyev, *Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them*. USA patent 7,860,914, 2010.

[16] M. Cococcioni, M. Pappalardo, and Y. D. Sergeyev, "Towards lexicographic multi-objective linear programming using grossone methodology," in *Proc. of the 2nd Intern. Conf. "Numerical Computations: Theory and Algorithms"* (S. Y. D., K. D. E., D. F., and M. M. S., eds.), vol. 1776, p. 090040, New York: AIP Publishing, 2016.

[17] L. Lai, L. Fiaschi, and M. Cococcioni, "Solving mixed pareto-lexicographic many-objective optimization problems: The case of priority levels," *submitted to Swarm and Evolutionary Computation*, 2019.

[18] R. De Leone, G. Fasano, and Y. D. Sergeyev, "Planar methods and grossone for the conjugate gradient breakdown in nonlinear programming," *Computational Optimization and Applications*, vol. 71, pp. 73–93, 2018.

[19] S. De Cosmis and R. De Leone, "The use of grossone in mathematical programming and operations research," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8029–8038, 2012.

[20] R. De Leone, "Nonlinear programming and grossone: Quadratic programming and the role of constraint qualifications," *Applied Mathematics and Computation*, vol. 318, pp. 290–297, 2018.

[21] M. Gaudioso, G. Giallombardo, and M. S. Mukhametzhanov, "Numerical infinitesimals in a variable metric method for convex nonsmooth optimization," *Applied Mathematics and Computation*, vol. 318, pp. 312–320, 2018.

[22] Y. D. Sergeyev, D. E. Kvasov, and M. S. Mukhametzhanov, "On strong homogeneity of a class of global optimization algorithms working with infinite and infinitesimal scales," *Communications in Nonlinear Science and Numerical Simulation*, vol. 59, pp. 319–330, 2018.

[23] F. Caldarola, "The Sierpinski curve viewed by numerical computations with infinities and infinitesimals," *Applied Mathematics and Computation*, vol. 318, pp. 321–328, 2018.

[24] Y. D. Sergeyev, "Numerical point of view on Calculus for functions assuming finite, infinite, and infinitesimal values over finite, infinite, and infinitesimal domains," *Nonlinear Analysis Series A: Theory, Methods & Applications*, vol. 71, no. 12, pp. e1688–e1707, 2009.

[25] Y. D. Sergeyev, "Numerical infinities applied for studying Riemann series theorem and Ramanujan summation," in *AIP Conference Proceedings of ICNAAM 2017*, vol. 1978, p. 020004, New York: AIP Publishing, 2018.

[26] A. Zhigljavsky, "Computing sums of conditionally convergent and divergent series using the concept of grossone," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8064–8076, 2012.

[27] F. Caldarola, "The exact measures of the Sierpinski d-dimensional tetrahedron in connection with a diophantine nonlinear system," *Communications in Nonlinear Science and Numerical Simulation*, vol. 63, pp. 228–238, 2018.

[28] L. D'Alotto, "A classification of two-dimensional cellular automata using infinite computations," *Indian Journal of Mathematics*, vol. 55, pp. 143–158, 2013.

[29] Y. D. Sergeyev, "Evaluating the exact infinitesimal values of area of Sierpinski's carpet and volume of Menger's sponge," *Chaos, Solitons & Fractals*, vol. 42, no. 5, pp. 3042–3046, 2009.

[30] Y. D. Sergeyev, "Using blinking fractals for mathematical modelling of processes of growth in biological systems," *Informatica*, vol. 22, no. 4, pp. 559–576, 2011.

[31] Y. D. Sergeyev, "The exact (up to infinitesimals) infinite perimeter of the Koch snowflake and its finite area," *Communications in Nonlinear Science and Numerical Simulation*, vol. 31, no. 1–3, pp. 21–29, 2016.

[32] D. Iudin, Y. D. Sergeyev, and M. Hayakawa, "Interpretation of percolation in terms of infinity computations," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8099–8111, 2012.

[33] D. Iudin, Y. D. Sergeyev, and M. Hayakawa, "Infinity computations in cellular automaton forest-fire model," *Communications in Nonlinear Science and Numerical Simulation*, vol. 20, no. 3, pp. 861–870, 2015.

[34] M. Margenstern, "Fibonacci words, hyperbolic tilings and grossone," *Communications in Nonlinear Science and Numerical Simulation*, vol. 21, no. 1–3, pp. 3–11, 2015.

[35] Y. D. Sergeyev, "Counting systems and the First Hilbert problem," *Nonlinear Analysis Series A: Theory, Methods & Applications*, vol. 72, no. 3-4, pp. 1701–1708, 2010.

[36] Y. D. Sergeyev and A. Garro, "Single-tape and multi-tape Turing machines through the lens of the Grossone methodology," *Journal of Supercomputing*, vol. 65, no. 2, pp. 645–663, 2013.

[37] L. Fiaschi and M. Cococcioni, "Numerical asymptotic results in game theory using Sergeyev's Infinity Computing," *Int. Journal of Unconventional Computing*, vol. 14, no. 1, pp. 1–25, 2018.

[38] L. Fiaschi and M. Cococcioni, "Generalizing pure and impure iterated prisoners dilemmas to the case of infinite and infinitesimal quantities," in *In Proc. of the the 3rd International Conference and Summer School on Numerical Computations: Theory and Algorithms*, June 2019.

[39] L. Fiaschi and M. Cococcioni, "Non-archimedean game theory: a numerical approach," *in preparation*, 2019.

[40] D. Rizza, "A study of mathematical determination through Bertrand's Paradox," *Philosophia Mathematica*, vol. 26, no. 3, pp. 375–395, 2018.

[41] D. Rizza, "Numerical methods for infinite decision-making processes," *Int. Journal of Unconventional Computing*, vol. 14, no. 2, pp. 139–158, 2019.

[42] C. S. Calude and M. Dumitrescu, "Infinitesimal probabilities based on grossone," *SN Computer Science*, 2020.

[43] P. Amodio, F. Iavernaro, F. Mazzia, M. S. Mukhametzhanov, and Y. D. Sergeyev, "A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmetic," *Mathematics and Computers in Simulation*, vol. 141, pp. 24–39, 2017.

[44] Y. D. Sergeyev, "Higher order numerical differentiation on the Infinity Computer," *Optimization Letters*, vol. 5, no. 4, pp. 575–585, 2011.

[45] Y. D. Sergeyev, "Solving ordinary differential equations by working with infinitesimals numerically on the Infinity Computer," *Applied Mathematics and Computation*, vol. 219, no. 22, pp. 10668–10681, 2013.

[46] Y. D. Sergeyev, M. S. Mukhametzhanov, F. Mazzia, F. Iavernaro, and P. Amodio, "Numerical methods for solving initial value problems on the Infinity Computer," *Int. Journal of Unconventional Computing*, vol. 12(1), pp. 3–23, 2016.

[47] F. Iavernaro, F. Mazzia, M. S. Mukhametzhanov, and Y. D. Sergeyev, "Conjugate-symplecticity properties of Euler-Maclaurin methods and their implementation on the Infinity Computer," *Applied Numerical Mathematics*, 2020.

[48] M. Pappalardo and M. Passacantando, *Ricerca Operativa*. Pisa University Press, 2012.

[49] Y. D. Sergeyev, "Lagrange Lecture: Methodology of numerical computations with infinities and infinitesimals," *Rendiconti del Seminario Matematico dell'Università e del Politecnico di Torino*, vol. 68, no. 2, pp. 95–113, 2010.

[50] Y. D. Sergeyev, "Un semplice modo per trattare le grandezze infinite ed infinitesime," *Matematica nella Società e nella Cultura: Rivista della Unione Matematica Italiana*, vol. 8, no. 1, pp. 111–147, 2015.

[51] Y. D. Sergeyev, *Arithmetic of Infinity*. CS: Edizioni Orizzonti Meridionali, 2003, 2nd ed. 2013.