Optimization Letters
https://doi.org/10.1007/s11590-020-01644-6

ORIGINAL PAPER

The Big-M method with the numerical infinite *M*

Marco Cococcioni[1] · Lorenzo Fiaschi[1]

Received: 30 March 2020 / Accepted: 8 September 2020

Abstract
Linear programming is a very well known and deeply applied field of optimization theory. One of its most famous and used algorithms is the so called Simplex algorithm, independently proposed by Kantorovič and Dantzig, between the end of the 30s and the end of the 40s. Even if extremely powerful, the Simplex algorithm suffers of one initialization issue: its starting point must be a feasible basic solution of the problem to solve. To overcome it, two approaches may be used: the two-phases method and the Big-M method, both presenting positive and negative aspects. In this work we aim to propose a non-Archimedean and non-parametric variant of the Big-M method, able to overcome the drawbacks of its classical counterpart (mainly, the difficulty in setting the right value for the constant M). We realized such extension by means of the novel computational methodology proposed by Sergeyev, known as *Grossone Methodology*. We have validated the new algorithm by testing it on three linear programming problems.

Keywords Big-M method · Grossone methodology · Infinity computer · Linear programming · Non-Archimedean numerical computing

1 Introduction

Linear Programming (LP) is a branch of optimization theory which studies the minimization (or maximization) of a linear objective function, subject to linear equality and/or linear inequality constraints. LP has found a lot of successful applications both in theoretical and real world contexts, especially in countless engineering applications. Most of the algorithms developed so far to solve LP problems fall in one of the two following categories: basis exchange methods [19] and interior point methods [27].

✉ Lorenzo Fiaschi
lorenzo.fiaschi@phd.unipi.it

Marco Cococcioni
marco.cococcioni@unipi.it

1 Department of Information Engineering, University of Pisa, Pisa, Italy

Researchers argued a lot about the pros and the cons of these approaches, ending up asserting the equal dignity of both. The best choice depends on the characteristics of the problem at hand [16].

Concerning this work, we decided to focus on the class of basis exchange methods and, in particular, on its probably most famous member: the Simplex algorithm, firstly proposed in 1939 by Kantorovič [17] and independently rediscovered by Dantzig during the forties [8–10].

This algorithm needs a feasible basic solution to start from, an input which is not trivial to generate for complex real-world problems. The main ways to overcome this difficulty are: the two-phases approach [10] and the Big-M method [2,8,26].

The former splits the optimization in two-phases, and in each it runs the Simplex algorithm on a phase-specific problem. In the first one, the problem is a simple task for which a feasible starting point is known, and whose optimum is a valid basic solution for the original problem. The second one is the original problem itself. In spite of the guarantee of convergence to the optimum (if it exists), such approach wastes a lot of time, having to run the Simplex algorithm twice rather than once.

The other possibility is diametrically opposed to the previous one, indeed, it requires just one run of the Simplex algorithm, though on a problem different from the original one. The latter approach is known as Big-M method, since it requires a big parameter M: it consists of applying the Simplex algorithm to a modified and parametrized version of the original problem for which a feasible initial basic solution is known. In particular, the big constant M (along with some auxiliary variables) is added to the original problem; a good tuning of such constant also guarantees that the original problem and the modified one share the same optimum. Thus, the drawback of the Big-M method is that it adds a new parameter, which also needs to be properly set: a too small value does not guarantee the convergence to the same optimum of the original problem, while a too big value may generate loss of precision and numerical instabilities.

The actual goal of this work is to overcome the Big-M method issues by means of a non-Archimedean modification of the original method, that is converting the LP problem at hand into a new one where the cost function involves infinite and infinitesimal quantities too. In order to realize such non-Archimedean extension, we exploited the Grossone Methodology proposed by Y.D. Sergeyev (see [22] and references therein), a powerful framework which lets one numerically manipulate infinite and infinitesimal quantities. Even more interestingly, algorithms involving the Grossone Methodology can run on a hardware computing engine, called *Infinity Computer* (see patent [20]), which is a completely new kind of supercomputer. Once opportunely modified, the new problem can be solved by a recent extension of the Simplex algorithm due to Cococcioni et al. [4], namely the *gross*-Simplex algorithm (in brief, G-Simplex). The latter is an improved Simplex algorithm able to solve non-Archimedean LP problems too. Together, the G-Simplex algorithm and the non-Archimedean modified problem give birth to the parameter-less procedure presented in this work, called by us the Infinitely-Big-M method (abbreviated I-Big-M), since it relies on the use of an *infinitely big* constant, which is the same for all the problems and thus does not have to be specified by the user.

The remaining of the letter is structured as follows: Sect. 2 provides an introduction to LP and Big-M method, Sect. 3 contains an overview of the Grossone Methodology and a description of the Gross-Simplex algorithm, Sect. 4 presents our non-Archimedean extension of the Big-M method, i.e., the Infinitely-Big-M method. In Sect. 5 we present three experiments we conducted exploiting the I-Big-M method to solve LP problems, while Sect. 6 is devoted to the conclusions.

2 The linear programming problems and the Big-M method

A possible representation of an LP problem, which may be a little complex at first sight, is shown in Eq. (1):

$$\begin{aligned} \min \quad & \mathbf{d}^T \mathbf{z} \\ \text{s.t.} \quad & \boldsymbol{\Delta}_{\text{le}}\, \mathbf{z} \leq \mathbf{b}_{\text{le}} \\ & \boldsymbol{\Delta}_{\text{eq}}\, \mathbf{z} = \mathbf{b}_{\text{eq}} \\ & \boldsymbol{\Delta}_{\text{ge}}\, \mathbf{z} \geq \mathbf{b}_{\text{ge}} \\ & \mathbf{z} \geq \mathbf{0} \end{aligned} \tag{1}$$

where $\mathbf{z} \in \mathbb{R}_+^{n_z}$ is the unknowns vector, $\mathbf{d} \in \mathbb{R}^{n_z}$ is the cost vector, $\boldsymbol{\Delta}_i \in \mathbb{R}^{m_i \times n}$ and $\mathbf{b}_i \in \mathbb{R}_+^{m_i}$ are the feasibility constraints, $i = le, eq, ge$. Here and throughout the whole work, bold symbols represent vectors and matrices, while non-bold ones refer to scalars.

As said, the Simplex algorithm requires a feasible basic solution to start from which may not be easy to find by hand. The Big-M method [2,8,26] tackles the problem transforming the original problem by adding some auxiliary variables which guarantee the existence of an initial feasible basic solution. However, not being part of the original problem, all of them must be equal to zero in the optimal solution of the modified problem. In order to guide the optimizer towards this annihilation, the Big-M method adds the auxiliary variables to the cost function and weights them with a big penalty M. The form of the modified problem is shown in Eq. (2):

$$\begin{aligned} \min \quad & M\mathbf{1}^T \mathbf{a} + \mathbf{d}^T \mathbf{z} \\ \text{s.t.} \quad & \boldsymbol{\Delta}_{\text{le}}\, \mathbf{z} + \mathbf{I}\mathbf{s} = \mathbf{b}_{\text{le}} \\ & \boldsymbol{\Delta}_{\text{eq}}\, \mathbf{z} + \mathbf{I}\mathbf{e} = \mathbf{b}_{\text{eq}} \qquad , \quad \mathbf{a} = \begin{bmatrix} \mathbf{e} \\ \mathbf{r} \end{bmatrix}, \\ & \boldsymbol{\Delta}_{\text{ge}}\, \mathbf{z} + \mathbf{I}\mathbf{r} - \mathbf{I}\mathbf{p} = \mathbf{b}_{\text{ge}} \\ & \mathbf{z}, \mathbf{s}, \mathbf{e}, \mathbf{p}, \mathbf{r} \geq \mathbf{0} \end{aligned} \tag{2}$$

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{3}$$

where $\mathbf{a} \in \mathbb{R}^{n_a}$ is the vector of auxiliary variables to be penalized ($n_a = m_{eq} + m_{ge}$), M is the penalty weight, $\mathbf{1}$ is a vector of n_a ones, $\mathbf{s}$ and $\mathbf{p}$ are m_{le} and m_{ge} slack variables, respectively. Such new form can be easily turned into the standard one (3), by applying the equalities shown in Eq. (4):

$$\mathbf{x} = \begin{bmatrix} \mathbf{z} \\ \mathbf{0} \\ \mathbf{a} \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \\ M\mathbf{1} \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \boldsymbol{\Delta}_{\text{le}} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Delta}_{\text{eq}} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Delta}_{\text{ge}} & \mathbf{0} & \mathbf{0} & \mathbf{I} & -\mathbf{I} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_{\text{le}} \\ \mathbf{b}_{\text{eq}} \\ \mathbf{b}_{\text{ge}} \end{bmatrix} \qquad (4)$$

where $\mathbf{x} \in \mathbb{R}_{+}^{n_x}$, $n_x = n_z + m$, $m = m_{le} + m_{eq} + m_{ge}$, $\mathbf{A} \in \mathbb{R}^{m \times n_x}$, $\mathbf{b} \in \mathbb{R}_{+}^{m}$. The always feasible initial basic solution for the modified problem is $\{\mathbf{s},\ \mathbf{e},\ \mathbf{r}\}$.

However, the Big-M method has a couple of relevant drawbacks. Actually, it turns a parameter-less problem into a parametric one, adding the scalar factor M. In addition, such parameter needs to be a-priory carefully set, in order to guarantee the correctness of the optimization process. Indeed, a too low value could lead to a non-zeroing of the auxiliary variables (and, thus, to no feasible solutions, even if they exist), a too high value could lead to a loss of precision and to numerical instabilities. Such setting issues, along with the need of re-tuning M for each different problem to solve, are crucial and struggling aspects from a practical point of view, sometimes limiting the applicability of the method in real contexts at all. In Sect. 4, we will present our parameter-free extension of the Big-M method which also preserves all its positive properties. With this goal in mind, let us move to the next section which introduces the methodological platform upon which to build such extension.

3 The Grossone methodology and the G-simplex algorithm

The Grossone Methodology (GM), which has been proved to stay apart from non-Standard Analysis [23], is a novel way for dealing with infinite, finite and infinitesimal numbers at once and in a numerical way [22]. Originally proposed in 2003, such framework already counts a lot of practical applications, for example it has been applied to non-linear optimization [11,18], global optimization [25], ordinary differential equations [1,15,21,24], control theory [3,12], and game theory [13,14], to cite a few. Also linear optimization positively enjoyed the advent of GM, as shown in [4–7]. In particular, in [4,6] the authors proposed an extension of the Simplex algorithm able to address Lexicographic Multi-Objective LP problems, both in absence and in presence of integer constraints, using a simplex-like approach. Such extension has been called *gross*-Simplex algorithm (briefly, G-Simplex), since it adopts GM to make the optimizer aware of the lexicographic priority between the objectives.

Actually, GM relates to the infinite unit *Grossone*, which is represented by the numeral ①. By means of it, a new numeral system with infinite base can be carried out, and each number $\tilde{c}$ within such numeral system can be represented as follows:

$$\tilde{c} = c_{p_m} ①^{p_m} + c_{p_{m-1}} ①^{p_{m-1}} + \cdots + c_0 ①^0 + \cdots + c_{p_{-k+1}} ①^{p_{-k+1}} + c_{p_{-k}} ①^{p_{-k}}. \qquad (5)$$

In Eq. (5) the number $\tilde{c}$ is called *gross*-scalar (in short, G-scalar), $m, k \in \mathbb{N}$, each p_i is called *gross*-power (G-power) and can be of the same form of $\tilde{c}$, while c_{p_i} is called *gross-digit* (G-digit) and is a finite (positive or negative) real number, $i = m, \ldots, -k$.

As mentioned, the G-Simplex is an algorithm which extends the Simplex procedure, being also able to solve the new class of LP problems known as *non-Archimedean* LP (NA-LP) problems. The latter have the canonical form described by Eq. (6):

$$\begin{aligned} \min \quad & \tilde{\mathbf{c}}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{6}$$

$$\begin{aligned} \min \quad & \sum_{i=1}^{l} \mathbf{c}_i^T\mathbf{x}①^{1-i} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{7}$$

where this time the cost vector $\tilde{\mathbf{c}}$ is a vector of non-Archimedean quantities, that is infinite and infinitesimal values, rather than just real numbers as in Eq. (3). In this work, we modeled $\tilde{\mathbf{c}}$ by means of a *gross*-vector (G-vector), i.e., an n-dimensional vector whose entries are G-scalars. In [4], it has been proved that a NA-LP problem is equivalent to the optimization of multiple linear functions among which a lexicographic priority relation exists, i.e., the problem expressed in Eq. (7) (l is the number of functions to optimize at the same time).

4 The Infinitely-Big-M method

The main issues related to the Big-M method concern the weight M. As said, it needs an a-priori careful settings because a too low value may not force the Simplex algorithm to nil the auxiliary variables, and a too big value may bring to loss of precision and numerical instabilities. Moreover, the weight needs to be opportunely tuned for each different LP problem, which means the task to solve is no more a parameter-less but a parametric one. Notwithstanding, the role of the weight M is just to prioritize the zeroing of the auxiliary variables over the optimization of $\mathbf{d}^T\mathbf{z}$, regardless its actual value. Indeed, it has just to put as much emphasis as possible on their minimization (reads to prioritize them), since a solution with at least one non-zero auxiliary variable is actually a non-feasible output for the original problem. Such priority lets us reformulate the modified problem into a non-Archimedean version.

The idea is to set the weight M equal to ①, in line with what De Cosmis and De Leone did in Nonlinear Programming [7]. Indeed, M represents a very big number which aims to increase the importance of the auxiliary variables zeroing over the minimization of $\mathbf{d}^T\mathbf{z}$. The canonical form of such modified problem is shown in Eq. (8).

$$
\begin{aligned}
\min \quad & \mathbf{1}^T \mathbf{a}① + \mathbf{d}^T \mathbf{z} \\
\text{s.t.} \quad & \mathbf{\Delta}_{\text{le}}\, \mathbf{z} + \mathbf{I}\mathbf{s} = \mathbf{b}_{\text{le}} \\
& \mathbf{\Delta}_{\text{eq}}\, \mathbf{z} + \mathbf{I}\mathbf{e} = \mathbf{b}_{\text{eq}} \\
& \mathbf{\Delta}_{\text{ge}}\, \mathbf{z} + \mathbf{I}\mathbf{r} - \mathbf{I}\mathbf{p} = \mathbf{b}_{\text{ge}} \\
& \mathbf{z}, \mathbf{s}, \mathbf{e}, \mathbf{p}, \mathbf{r} \geq \mathbf{0}
\end{aligned}
\tag{8}
$$

Imposing the equalities in Eq. (4) (except for the vector $\mathbf{c}$) and the ones in Eq. (9), the problem gets a form close to Eq. (7). Scaling down its cost function by a factor ①, it ens up fitting it perfectly. This means that such a problem can be solved by means of the G-Simplex algorithm. Indeed, as stated in Sect. 3, there always exists a bijection between a problem in the form of Eq. (7) and one in the form of Eq. (6). With regard to the Big-M method we aim to present in this letter, such bijection corresponds to Eq. (10).

$$
\mathbf{c}_1 = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{c}_2 = \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{9}
$$

$$
\tilde{\mathbf{c}} = \mathbf{c}_1① + \mathbf{c}_2 = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} ① + \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{10}
$$

Together, the rewriting of the modified problem into its non-Archimedean form and the use of the G-Simplex algorithm to solve it, implement the extended version of the Big-M method we sought to propose in this work. The latter is a procedure which leverages on an *infinitely* big constant to properly guide the optimization, which is the reason why we called it Infinitely-Big-M (in brief, I-Big-M). Notice that the I-Big-M method is a parameter-less approach, indeed, it does not require anymore the tuning of M, which is now *always* set to ①, regardless the problem to solve. Moreover, the issue of finding a feasible initial basic solution for the G-Simplex algorithm does not exist, since the modified problem and its non-Archimedean version are two different ways for representing the very same problem. What changes is only how to guide the optimization process, that is the loss function, while the polytope describing the feasible region stays the same. This means that a feasible initial basic solution for the Simplex algorithm in the modified problem is also a polytope vertex for the G-Simplex. Thus, Big-M and a I-Big-M methods share both their starting and optimal points (to be precise, the latter is true only if the weight M is correctly set in the Big-M method). Finally, it is worth to highlight how the I-Big-M method is able to tackle lexicographic multi-objective LP problems straightforwardly and without any modification to the method itself. Indeed, on the one hand the problem modification remains the same, this time with more than just one secondary vector. On the other hand, the G-Simplex algorithm is able to solve NA-LP problems regardless the number of objectives to deal with. Thus, the I-Big-M method, which is the combination of the problem modification and the G-Simplex algorithm, does not care whether the original problem to solve is lexicographic multi-objective or just single-objective. Numerical evidences of such property will be shown in Sect. 5.2.

5 A numerical illustration

In this section we show three practical cases of study where we can appreciate the effectiveness of the I-Big-M algorithm.

5.1 First case of study

The first analysed problem is a slight variant of Example 1 provided in [4]. Our variant takes into account only the first objective of that benchmark, which is actually a lexicographic multi-objective problem.

In order to involve an equality constraint, we have also introduced the additional variable z_3, which plays a very marginal role in the economy of the problem. The analytical form of the current case of study is shown below, while its graphical description is reported in Fig. 1 (after being projected onto the plane $z_3 = 10$).

$$\begin{aligned} \min \quad & -8z_1 - 12z_2 - 7z_3 \\ \text{s.t.} \quad & 2z_1 + z_2 - 3z_3 \leq 90 \\ & 2z_1 + 3z_2 - 2z_3 \leq 190 \\ & 4z_1 + 3z_2 + 3z_3 \leq 300 \\ & z_3 = 10 \\ & z_1 + 2z_2 + z_3 \geq 70 \\ & z_1, z_2, z_3 \geq 0 \end{aligned}$$

For completeness, below we also report the matrices $\mathbf{\Delta}_i$ and the vectors $\mathbf{b}_i$ by means of which we can represent the current LP problem in the form of Eq. (1), $i = le,\ eq,\ ge$:

$$\mathbf{\Delta}_{\text{le}} = \begin{bmatrix} 2 & 1 & -3 \\ 2 & 3 & -2 \\ 4 & 3 & 3 \end{bmatrix}, \quad \mathbf{b}_{\text{le}} = \begin{bmatrix} 90 \\ 190 \\ 300 \end{bmatrix}, \quad \begin{array}{l} \mathbf{\Delta}_{\text{eq}} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\ \mathbf{b}_{\text{eq}} = \begin{bmatrix} 10 \end{bmatrix} \end{array}, \quad \begin{array}{l} \mathbf{\Delta}_{\text{ge}} = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \\ \mathbf{b}_{\text{ge}} = \begin{bmatrix} 70 \end{bmatrix} \end{array} \tag{11}$$

The problem's optimum is degenerate: any point within the red segment of Fig. 1 is optimal. We have indicated as $\alpha = [0, 70, 10]$ and $\beta = [30, 50, 10]$ the two vertices of the optimal segment. In Table 1, we reported the iterations that the G-Simplex algorithm performed once run on our Infinity Computer simulator implemented in Matlab. As it can be seen from the table, the G-Simplex algorithm immediately gets rid of the auxiliary variables. Indeed, in the very first iteration the auxiliary variable r exits from the base, followed by variable e during the second iteration (in this case vectors $\mathbf{r}$ and $\mathbf{e}$ are actually scalars). This is confirmed by the cost function, which assumes a finite value in exactly two steps. Thus, the infinite weight ① does its own duty, that is prioritize the zeroing of the auxiliary variables during the solution of

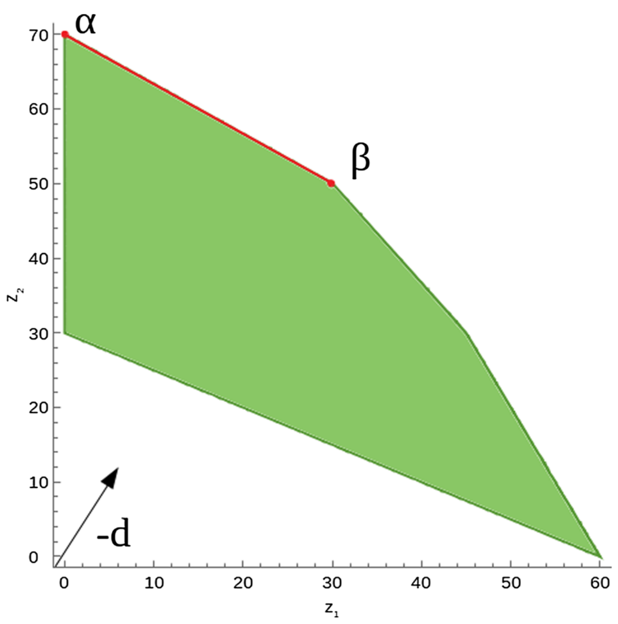

Fig. 1 Graphical representation of the first case of study in $\mathbb{R}^2$ (actually, along the plane $z_3 = 10$). The polygon is highlighted in green, while the degenerate optimal solutions in red. Vector $\mathbf{d}$ is the cost vector of the original problem: $\mathbf{d} = [-8, -12, -7]^T$. Here we have depicted the negation of its projection on the $z_1 z_2$-plane (color figure online)

Table 1 The Algorithm's iterations to find the optimum $\mathbf{z}^* = [0, 70, 10]$

Iter.	$\mathbf{x}^*$	$\tilde{\mathbf{c}}^T \mathbf{x}^*$	**Base**	$\mathbf{z}^*$
1	[0, 0, 0, 90, 190, 300, 10, 70, 0]	$80①^1 + 0①^0$	{4,5,6,7,8}	[0, 0, 0]
2	[0, 35, 0, 55, 85, 195, 10, 0, 0]	$10①^1 - 420①^0$	{4,5,6,7,2}	[0, 35, 0]
3	[0, 30, 10, 90, 120, 180, 0, 0, 0]	$0①^1 - 430①^0$	{4,5,6,3,2}	[0, 30, 10]
4	[0, 70, 10, 50, 0, 60, 0, 0, 0, 80]	$0①^1 - 910①^0$	{4,9,6,3,2}	**[0,70,10]**

the problem, which manifests in the auxiliary variables exiting from the optimal base before all the other unknowns. Finally, the optimum output by the routine was $\mathbf{z}^* = \alpha$.

5.2 Second case of study

The aim of this second case of study is to numerically verify that the I-Big-M method is able to tackle the lexicographic multi-objective optimization transparently with respect to the user and the problem modification procedure. To do it, we run the I-Big-M routine on the problem presented in Eq. (12) and graphically described in Fig. 2. The new problem is the same as before with the addition of two new linear cost functions. The three overall cost vectors are reported in Eq. (13), and they can be lexicographically ordered by priority as follows: $\mathbf{d} \gg \mathbf{d}_2 \gg \mathbf{d}_3$. We avoided to report the problem splittings in matrices $\mathbf{\Delta}_i$ and vectors $\mathbf{b}_i$ $(i = le,\ eq,\ ge)$, since the splitting is exactly the same already reported in Eq. (11):

$$\begin{aligned} \text{lexmin} \quad & \mathbf{d}^T\mathbf{z},\ \mathbf{d}_2^T\mathbf{z},\ \mathbf{d}_3^T\mathbf{z} \\ \text{s.t.} \quad & 2z_1 + \ \ z_2 - 3z_3 \leq 90 \\ & 2z_1 + 3z_2 - 2z_3 \leq 190 \\ & 4z_1 + 3z_2 + 3z_3 \leq 300 \\ & z_3 = 10 \\ & z_1 + 2z_2 + z_3 \geq 70 \\ & z_1, z_2, z_3 \geq 0 \end{aligned} \tag{12}$$

where

$$\mathbf{d} = \begin{bmatrix} -8 \\ -12 \\ -7 \end{bmatrix}, \quad \mathbf{d}_2 = \begin{bmatrix} -14 \\ -10 \\ -2 \end{bmatrix}, \quad \mathbf{d}_3 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \tag{13}$$

Some aspects of the problem are worth to be pointed out. Actually, the optimum of the new problem is now the point β, as higlighted by Fig. 2. Indeed, while for the first objective ($\mathbf{d}$) the problem solution is degenerate (both α and β are acceptable), the function $\mathbf{d}_2$ lets us discriminate between them. Thus, while before the choice between them was delegated to the G-Simplex algorithm actual implementation, now it is guided by the second objective function and it falls upon β. The third and least optimality direction, i.e., $\mathbf{d}_3$ plays no role, since the optimum is just a singleton this time.

Finally, in Table 2 we reported the iterations performed by the G-Simplex algorithm when solving the lexicographic multi-objective LP problem in Eq. (12). Notice that the first four iterations of the algorithm are exactly the same reported in Table 1. The reason behind this phenomenon is twofold. Firstly, the G-Simplex solves both the problems (this and the previous one) from the very same initial basic solution $\{\mathbf{s},\ \mathbf{e},\ \mathbf{a}\}$, chosen by the I-Big-M method. Secondly, the optimization is guided by the objective $\mathbf{d}$ until the degenerate solution is found. Together, this two aspects make reasonable to expect that the G-Simplex algorithm will follow the same steps in both the problems until it finds the degenerate solution. At that point, the first objective is not able anymore to discriminate among the solutions, and therefore the second objective turns to be effective, leading the algorithm towards the unique lexicographic solution of the problem β.

Before concluding, in Eq. (14) we report the non-Archimedean objective function of the modified NA-LP problem generated by the I-Big-M method. Such function lets us write such problem in the form of Eq. (6):

$$\tilde{\mathbf{c}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} ① + \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} ①^0 + \begin{bmatrix} \mathbf{d_2} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} ①^{-1} + \begin{bmatrix} \mathbf{d_3} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} ①^{-2} \tag{14}$$

Table 2 The Algorithm's iterations to find the optimum $\mathbf{z}^* = [35, 50, 10]$

Iter.	$\mathbf{x}^*$	$\tilde{\mathbf{c}}^T \mathbf{x}^*$	**Base**	$\mathbf{z}^*$
1	[0, 0, 0, 90, 190, 300, 10, 70, 0]	$80①^1 + 0①^0 + 0①^{-1} + 0①^{-2}$	{4,5,6,7,8}	[0, 0, 0]
2	[0, 35, 0, 55, 85, 195, 10, 0, 0]	$10①^1 - 420①^0 - 350①^{-1} - 35①^{-2}$	{4,5,6,7,2}	[0, 35, 0]
3	[0, 30, 10, 90, 120, 180, 0, 0, 0]	$0①^1 - 430①^0 - 320①^{-1} - 40①^{-2}$	{4,5,6,3,2}	[0, 30, 10]
4	[0, 70, 10, 50, 0, 60, 0, 0, 0, 80]	$0①^1 - 910①^0 - 720①^{-1} - 80①^{-2}$	{4,9,6,3,2}	[0, 70, 10]
5	[30, 50, 10, 10, 0, 0, 0, 0, 0, 70]	$0①^1 - 910①^0 - 940①^{-1} - 90①^{-2}$	{4,9,1,3,2}	**[30,50,10]**

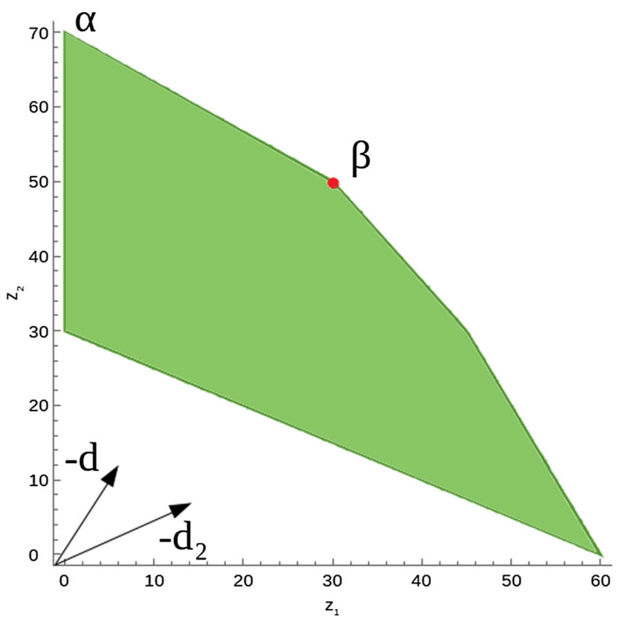

Fig. 2 Graphical representation of the second case of study assuming $z_3 = 10$. The polygon is highlighted in green, while the optimal solution $\mathbf{z}^* = [35, 50, 10]$ (β) in red. $\mathbf{d}_3$ has not been drawn, since it does not play any role (color figure online)

5.3 Third case of study

We now illustrate a third case of study where the traditional Big-M method struggles, because setting the right value for M is not trivial. We have considered the problem `neos-1425699`, which can be downloaded from: https://miplib.zib.de/instance_details_neos-1425699.html

The problem is included within the MIPLIB2017 collection of Mixed Integer LP benchmarks, and it contains 63 inequality constraints and 26 equality constraints. Of course, we have removed the integrality constraint from all the 105 variables, and we have also removed their upper bounds. As it can be seen from Table 3, the known optimal value of the cost function is $\mathbf{3.14867 \cdot 10^9}$ (found using Matlab `linprog` function). Using values for M below 10^8 the standard Big-M algorithm is not able to find the right optimum, as proved by a different value of the cost function. Also the number of iterations performed by the Big-M method is interesting: it passes from 26 to 89 (see the third column of Table 3). Needless to say, on the same problem the I-Big-M method is able to find the optimum at the first attempt, i.e., without any trial-and-error approach. Thus, we think that the I-Big-M method can be a powerful method in those setting which require the solution of LP problems as a sub-problem, like in the case of Mixed Integer Linear Programming when approached using a Branch-and-Bound/Branch-and-Cut strategy, both for the single objective and the lexicographic multiple-objective cases [6].

Table 3 An example of problem where setting the right value of M in the standard Big-M method is not trivial

Problem: **neos-1425699.mps**
(known optimal value of the cost function: $\mathbf{3.14867 \cdot 10^9}$)

Value of M	Optimum cost function found by Big-M	Number of iterations
10^2	$1.36403 \cdot 10^7$	26
10^3	$1.29867 \cdot 10^8$	26
10^4	$1.25666 \cdot 10^9$	30
10^5	$2.95207 \cdot 10^9$	88
10^6	$2.95535 \cdot 10^9$	88
10^7	$2.98818 \cdot 10^9$	88
10^8	$\mathbf{3.14867 \cdot 10^9}$	89
10^9	$\mathbf{3.14867 \cdot 10^9}$	89

6 Conclusions

In this letter we have introduced a new algorithm, called I-Big-M method, which is a re-visitation of the Big-M method for the Infinity Computer. The I-Big-M method is able to solve a LP problem using only one run of the G-Simplex algorithm, in the very same manner of the classical Big-M method, which uses a single run of the classical Simplex algorithm. Even more interesting, the I-Big-M method does not require the constant M, a parameter difficult to set, typically chosen by an error-prone trial-and-error approach. Hence, the I-Big-M offers a parameter-less algorithm to solve LP problems, generating a single sequence of visited vertices. Moreover, its formulation also allows one to solve lexicographic multi-objective LP problems in a natural way, again without any assistance by the user and solving the problem using just a single run of the G-Simplex algorithm (i.e., by generating overall a single sequence of visited vertices: a pretty remarkable result). All these properties have been numerically validated by means of three experiments, one of which involving a real-world challenging benchmark. Finally, the I-Big-M method is expected to be more numerically stable than the original Big-M method, since it does not mix numbers with very different order of magnitudes. The verification of the latter conjecture deserves a deeper investigation and it is left to a future study.

Acknowledgements This work has been partially supported by the University of Pisa funded project PRA_2018_81 “Wearable sensor systems: personalized analysis and data security in healthcare”, and partially by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). We also thank the three anonymous reviewers for their valuable remarks.

Funding Open access funding provided by Università di Pisa within the CRUI-CARE Agreement.

References

1. Amodio, P., Iavernaro, F., Mazzia, F., Mukhametzhanov, M., Sergeyev, Y.D.: A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmetic. Math. Comput. Simul. **141**, 24–39 (2017). https://doi.org/10.1016/j.matcom.2016.03.007
2. Bazaraa, M.S., Jarvis, J.J., Sherali, H.D.: Linear Programming and Network Flows. Wiley, New York (1990)
3. Caldarola, F., Maiolo, M., Solferino, V.: A new approach to the z-transform through infinite computation. Commun. Nonlinear Sci. Numer. Simul. **82**, 105019 (2020). https://doi.org/10.1016/j.cnsns.2019.105019
4. Cococcioni, M., Pappalardo, M., Sergeyev, Y.D.: Lexicographic multi-objective linear programming using grossone methodology: theory and algorithm. Appl. Math. Comput. **318**, 298–311 (2018). https://doi.org/10.1016/j.amc.2017.05.058
5. Cococcioni, M., Cudazzo, A., Pappalardo, M., Sergeyev, Y.D.: Grossone methodology for lexicographic mixed-integer linear programming problems. In: International Conference on Numerical Computations: Theory and Algorithms, pp. 337–345. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-40616-5_28
6. Cococcioni, M., Cudazzo, A., Pappalardo, M., Sergeyev, Y.D.: Solving the lexicographic multi-objective mixed-integer linear programming problem using branch-and-bound and grossone methodology. Communications in Nonlinear Science and Numerical Simulation, pp. 105–177 (2020). https://doi.org/10.1016/j.cnsns.2020.105177
7. De Cosmis, S., De Leone, R.: The use of grossone in mathematical programming and operations research. Appl. Math. Comput. **218**(16), 8029–8038 (2012). https://doi.org/10.1016/j.amc.2011.07.042
8. Dantzig, G.B.: Programming in a Linear Structure. Comptroller, United Air Force, Washington, D.C., Tech rep (1948)
9. Dantzig, G.B., Thapa, M.N.: Linear Programming 1: Introduction. Springer, New York (1997)
10. Dantzig, G.B., Thapa, M.N.: Linear Programming 2: Theory and Extensions. Springer, New York (2003)
11. De Leone, R.: Nonlinear programming and grossone: quadratic programming and the role of constraint qualifications. Appl. Math. Comput. **218**(16), 290–297 (2018). https://doi.org/10.1016/j.amc.2017.03.029
12. Falcone, A., Garro, A., Mukhametzhanov, M.S., Sergeyev, Y.D.: A simulink-based infinity computer simulator and some applications. In: International Conference on Numerical Computations: Theory and Algorithms, pp. 362–369. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-40616-5_31
13. Fiaschi, L., Cococcioni, M.: Numerical asymptotic results in game theory using Sergeyev's arithmetic of infinity. Int. J. Unconv. Comput. **14**, 1–25 (2018)
14. Fiaschi, L., Cococcioni, M.: Non-Archimedean game theory: a numerical approach. Appl. Math. Comput. (2020). https://doi.org/10.1016/j.amc.2020.125356
15. Iavernaro, F., Mazzia, F., Mukhametzhanov, M., Sergeyev, Y.D.: Conjugate-symplecticity properties of euler-maclaurin methods and their implementation on the infinity computer. Appl. Numer. Math. (2019). https://doi.org/10.1016/j.apnum.2019.06.011
16. Illés, T., Terlaky, T.: Pivot versus interior point methods: pros and cons. Eur. J. Oper. Res. **140**(2), 170–190 (2002). https://doi.org/10.1016/S0377-2217(02)00061-9
17. Kantorovič, L.V.: Mathematical methods of organizing and planning production. Publ. House Leningrad State Univ. (1939). https://doi.org/10.1287/mnsc.6.4.366
18. Lai, L., Fiaschi, L., Cococcioni, M.: Solving mixed Pareto-lexicographic multi-objective optimization problems: the case of priority chains. Swarm and Evolutionary Computation, p. 100687 (2020). https://doi.org/10.1016/j.swevo.2020.100687
19. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Courier Corporation (1998)

20. Sergeyev, Y.D.: Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them. USA patent 7,860,914 (2010)
21. Sergeyev, Y.D.: Solving ordinary differential equations by working with infinitesimals numerically on the infinity computer. Appl. Math. Comput. **219**(22), 10668–10681 (2013). https://doi.org/10.1016/j.amc.2013.04.019
22. Sergeyev, Y.D.: Numerical infinities and infinitesimals: methodology, applications, and repercussions on two Hilbert problems. EMS Surv. Math. Sci. **4**, 219–320 (2017). https://doi.org/10.4171/EMSS/4-2-3
23. Sergeyev, Y.D.: Independence of the grossone-based infinity methodology from non-standard analysis and comments upon logical fallacies in some texts asserting the opposite. Found. Sci. **24**(1), 153–170 (2019). https://doi.org/10.1007/s10699-018-9566-y
24. Sergeyev, Y.D., Mukhametzhanov, M., Mazzia, F., Iavernaro, F., Amodio, P.: Numerical methods for solving initial value problems on the infinity computer. Int. J. Unconv. Comput. **12**(1), 3–23 (2016)
25. Sergeyev, Y.D., Kvasov, D.E., Mukhametzhanov, M.S.: On strong homogeneity of a class of global optimization algorithms working with infinite and infinitesimal scales. Commun. Nonlinear Sci. Numer. Simul. **59**, 319–330 (2018). https://doi.org/10.1016/j.cnsns.2017.11.013
26. Soleimani-damaneh, M.: Modified big-m method to recognize the infeasibility of linear programming models. Knowl.-Based Syst. **21**(5), 377–382 (2008). https://doi.org/10.1016/j.knosys.2008.02.004
27. Terlaky, T.: Interior Point Methods of Mathematical Programming, vol. 5. Springer, Berlin (2013)