A genetic algorithm for optimization over a simplex

Carlos H. Dias[a], Francisco A. M. Gomes[*,a]

[a]*Departamento de Matemática Aplicada, Universidade Estadual de Campinas, Campinas, SP, Brazil*

Abstract

Several combinatorial optimization problems require the minimization of a function over the standard n-dimensional simplex $x_1 + x_2 + \ldots + x_n,\ x \geq 0$. Due to the high complexity of these problems, it is a common practice to solve them using metaheuristics, such as evolutionary algorithms. In this paper, we present a genetic algorithm based on a new chromosome representation that allows the direct generation of feasible solutions. With this representation, the solution space is discretized into hypercubes, and each hypercube is associated to an unique integer number that is stored in binary form to simplify the definition of the crossover and mutation operators.

As an example, we use our approach to find the efficient frontier of some cardinality constrained portfolio optimization problems. Our experiments show that the new representation gives better results than the chromosomes built using the Cartesian coordinates on $\mathbb{R}^n$.

Key words: genetic algorithms, simplex constraints, portfolio optimization
2000 MSC: 90C59, 65K05

1. Introduction

In this paper we consider optimization problems in which the variables are restricted to the intersection of an hyperplane in $\mathbb{R}^n$ with the nonnegative orthant, i.e., the feasible region is given by the $(n-1)$-dimensional simplex

$$\Omega = \left\{ x \in \mathbb{R}^n | \sum_{i=1}^{n} c_i x_i,\ x_i \geq 0, i = 1, \ldots, n \right\}, \tag{1}$$

where $c = [c_1\ c_2 \ldots c_n]^T$ is a constant vector.

Several problems have constraints in the form (1). The application we are most concerned is the cardinality constrained portfolio optimization. In this problem, a set of m assets (such as stocks and bonds) is available and one must

*Corresponding author
Email address: chico@ime.unicamp.br (Francisco A. M. Gomes)

select some of them to compose a portfolio that gives the highest expected return with minimum risk. When the Markowitz model [11] is used, the best portfolio is the solution of the quadratic programming problem

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m}\sigma_{i,j}w_i w_j \\ \textit{subject to} \quad & \sum_{i=1}^{m} r_i w_i = R, \\ & \sum_{i=1}^{m} w_i = 1, \\ & w_i \geq 0,\ i = 1,\ldots,m, \end{aligned} \tag{2}$$

where w_i is the fraction of the portfolio value invested in asset i, σ_{ij} is the covariance between assets i and j, r_i is the expected return of asset i and R is the desired level of return for the whole portfolio. Other models and risk measures can be found in [12]

Model (2) may be unrealistic, since it permits the investor to buy an arbitrarily small amount of a large number of different assets. To reduce transaction costs and allow an accurate control of the invested capital, it is a common practice to limit the number of assets held in the portfolio, as well as to define a buy-in threshold, i.e., a lower limit for the amount invested on each asset. Unfortunately, introducing these constraints into (2), we get the following mixed integer quadratic programming problem (MIQP)

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{m}\sigma_{i,j}w_i w_j \\ \textit{subject to} \quad & \sum_{i=1}^{m} r_i w_i = R, \\ & \sum_{i=1}^{m} w_i = 1, \\ & \sum_{i=1}^{m} z_i = n, \\ & l_i z_i \leq w_i \leq u_i z_i,\ i = 1,\ldots,m, \\ & z_i \in \{0,1\},\ i = 1,\ldots,m. \end{aligned} \tag{3}$$

This problem can be slightly simplified if we remove the first constraint and trade risk against return directly in the objective function. In this case, we suppose that a trade-off parameter $\lambda \in [0,1]$ is given and rewrite (3) as

$$\textit{minimize} \quad \lambda\sum_{i=1}^{m}\sum_{j=1}^{m}\sigma_{i,j}w_i w_j - (1-\lambda)\sum_{i=1}^{m} r_i w_i$$

$$
\begin{aligned}
\textit{subject to} \quad & \sum_{i=1}^{m} w_i = 1, \\
& \sum_{i=1}^{m} z_i = n, \\
& l_i z_i \leq w_i \leq u_i z_i, \ i = 1, \ldots, m, \\
& z_i \in \{0, 1\}, \ i = 1, \ldots, m.
\end{aligned} \tag{4}
$$

Since (4) is known to be an NP-hard problem, several metaheuristics were proposed for solving it, including genetic algorithms [3], simulated annealing [3], the combination of simulated annealing and evolutionary strategies [10], tabu search [3], neural networks [7], particle swarm optimization (without the cardinality constraints, but including transaction costs and tax) [18]. The combination of metaheuristics and quadratic programming was explored in [6] and [13]. Methods based purely on nonlinear programming ideas were also proposed for solving (3), including some variations of the branch-and-bound method [2, 16, 8] and DC programming [9]. Finally, in [1] problem (3) was converted into an unconstrained nonconvex nonlinear programming problem, and solved by a global optimization procedure.

In this paper, we introduce a new chromosome representation scheme that improves the efficiency of genetic algorithms applied to solve problem with simplex constraints such as (1). The paper is structured as follows. In the next section, we present the chromosome representation usually adopted when solving mixed integer problems with simplex constraints. In Section 3, we introduce our chromosome representation. A simple genetic algorithm that uses the new chromosome for solving (3) is presented in Section 4. A simplified version of the algorithm is given in Section 5. The new algorithm is compared to the one proposed by Chang *et al.*[3] in Section 6. Finally, some conclusions are presented in Section 6, along with suggestions for future work.

2. Usual chromosome representation for simplex constrained mixed integer problems

The first step in the definition of a genetic algorithm is the decision on how a feasible solution will be coded into a chromosome. For mixed integer cardinality constrained problems such as (4), usually the chromosome is divided into two parts (see, for example, [3]). The first is an integer vector I that stores the indices of the n assets that belong to the portfolio, i.e., the indices of the components of z that are equal to 1. The second part is a vector W that stores the values of w_i, $i \in I$.

When there are no upper limits for the investment on the assets, i.e., $u_i = 1$, $i = 1, \ldots, n$, there is a simple way to convert the constraints of (4) into the feasible region Ω.

To make this conversion, let us define $L = \sum_{j \in I} l_j$. Once the indices of assets that belong to the portfolio are stored in vector I, the constraints of (4)

reduce to

$$\sum_{i \in I} w_i = 1, \tag{5}$$

$$l_i \leq w_i \leq 1 - L + l_i, \ i \in I. \tag{6}$$

Now, if we define a new vector x such that

$$w_i = l_i + (1 - L)x_i, \ i \in I,$$

then (5)–(6) may be rewritten as

$$\sum_{i \in I} x_i = 1, \quad x_i \geq 0, \ i \in I. \tag{7}$$

Therefore, if there are no upper limits on w, the only constraint we must consider when applying a genetic algorithm to solve (3) is (7). However, defining a chromosome representation well suited for dealing with this constraint may be tricky.

Most authors use the real part of the chromosome to store a point x that belongs to the unit hypercube in $\mathbb{R}^n$. Since this point may be unfeasible, a common practice to map x to a point x' on Δ_{n-1} is to project x onto Δ_{n-1} along the ray that passes through the point and the origin (see, for example, [3, 15, 14, 4]). Unfortunately, if we take p points uniformly distributed on the unit hypercube and project them onto the simplex, the generated points will not be evenly distributed, as shown in Figure 1.

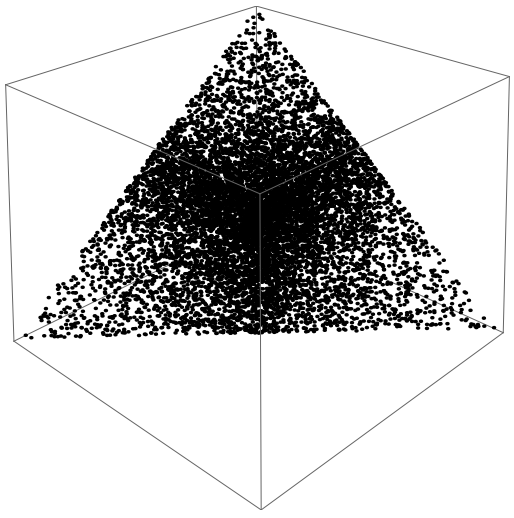

Figure 1: The uneven distribution of points obtained projecting 10,000 points of the unit hypercube onto the simplex.

Naturally, using a cleverer scheme, it is possible to generate an initial population that is uniformly distributed on the simplex[1]. However, if we use the Cartesian coordinates to store the real part of a chromosome, the crossover and the mutation operators will usually destroy the feasibility of the solution, so it will be necessary to re-project the point onto the simplex after applying these procedures, increasing the concentration of chromosomes on certain portions of the simplex. In the next section, we propose a chromosome representation that avoids this inconvenience.

3. A new chromosome representation

We want to map directly each chromosome to a point of the simplex in such a way that this mapping allows an uniform distribution of the points over the simplex.

If d bits are used to represent a real number on a computer, then it would be necessary to use dn bits to store a vector $x \in \mathbb{R}^n$. We intend to use the same amount of memory to store a point on the standard simplex, associating a single natural number y between 1 and 2^{dn} to this point.

To do so, we divide the $(n-1)$-dimensional simplex Δ_{n-1} into 2^{dn} hypercubes, each one represented by its center. Although the simplex contains infinite points that will not be addressed by this representation and the hypercubes may include points that do not belong to the simplex, the inaccuracy of this scheme is negligible as long as we use a huge number of hypercubes. In fact, our representation is even more precise than the one obtained using the same number of bits to store the Cartesian coordinates of a point on the simplex.

On the following subsections, we explain how to obtain the coordinates of the center of a hypercube, once given $y \in \{1, \ldots, 2^{dn}\}$, the index of this hypercube on Δ_{n-1}. We start presenting a few facts about simplexes. After that, we discuss how the hypercubes can be accommodated into layers. Then we show how to compute the layers associated to a single hypercube. Finally, we convert the layers into Cartesian coordinates.

3.1. Content and height of a regular simplex

The feasible region Ω can be viewed as the convex hull of the linearly independent points $p_i = c_i e_i \in \mathbb{R}^n,\ i = 1, \ldots, n$, where e_i is the i-th column of the $n \times n$ identity matrix. If $c = \alpha[1\ 1 \ldots 1]^T$, $\alpha \in \mathbb{R}$, the simplex is called *regular*. Moreover, if $\alpha = 1$, we have the *standard* (or *unit*) $(n-1)$-dimensional simplex, represented by Δ_{n-1}.

The *content* (or volume) of an $(n-1)$-dimensional simplex is given by the Cayley-Menger determinant (see, for example, [17, p. 363]). If the simplex is

[1] To generate a point on the $(n-1)$-dimensional simplex, one should generate $(n-1)$ real numbers in $[0, 1]$, sort them, and compute the difference between each pair of successive numbers, including the 0 and the 1 as the first and the last numbers of the list. More information on this strategy for computing points on the simplex can be found in [5].

regular, with edges of length s, the formula of its content reduces to

$$V_{n-1}(s) = \frac{s^{n-1}}{(n-1)!}\sqrt{\frac{n}{2^{n-1}}}. \tag{8}$$

For the standard simplex, the content is simply

$$V_{n-1} \equiv V_{n-1}(\sqrt{2}) = \frac{\sqrt{n}}{(n-1)!}. \tag{9}$$

Let $\Delta_{n-1}(s)$ be a regular simplex with edges of length s. We define the *height* of this simplex, and denote by $H_{n-1}(s)$, the Euclidean distance between one of its vertices and the opposite face. The following lemma provides a formula for the height of $\Delta_{n-1}(s)$.

Lemma 1. *The height of a regular $(n-1)$-dimensional simplex with edges of length s is given by*

$$H_{n-1}(s) = s\sqrt{\frac{n}{2(n-1)}}. \tag{10}$$

Proof. Suppose that $\Delta_{n-1}(s)$ has one vertex (called the *reference* vertex) at the origin, and the opposite face (the *basis* of the simplex) on a hyperplane that is orthogonal to the x-axis. Clearly, any cross section of this simplex that is parallel to its basis is an $(n-2)$-dimensional regular simplex. Let $\bar{\Delta}_{n-2}(s,x)$ be the $(n-2)$-dimensional simplex obtained sectioning $\Delta_{n-1}(s)$ at a distance x from the reference vertex. The length of the edges of $\bar{\Delta}_{n-2}(s,x)$ is given by

$$\bar{s} = \frac{x}{H_{n-1}(s)}s.$$

Besides, the content of $\bar{\Delta}_{n-2}(s,x)$ is

$$V_{n-2}(\bar{s}) = \left[\frac{\bar{s}}{s}\right]^{n-2} V_{n-2}(s) = x^{n-2}\left[\frac{V_{n-2}(s)}{H_{n-1}(s)^{n-2}}\right].$$

Therefore, we can write the content of the $(n-1)$-dimensional simplex $\Delta_{n-1}(s)$ as

$$V_{n-1}(s) = \int_0^{H_{n-1}(s)} x^{n-2}\left[\frac{V_{n-2}(s)}{H_{n-1}(s)^{n-2}}\right]dx = \frac{H_{n-1}(s)}{n-1}V_{n-2}(s). \tag{11}$$

Now, from (11) and (8), we obtain

$$H_{n-1}(s) = (n-1)\frac{V_{n-1}(s)}{V_{n-2}(s)} = s\sqrt{\frac{n}{2(n-1)}}.$$

□

To simplify the notation, let us denote the height of the standard $(n-1)$-dimensional simplex by H_{n-1}. Using (10), we deduce that

$$H_{n-1} = \sqrt{\frac{n}{n-1}}. \tag{12}$$

3.2. Simplex layers

Let us divide Δ_{n-1} into 2^{dn} $(n-1)$-dimensional hypercubes in such a way that the set of hypercubes and the simplex have the same content. In this case, the edges of each hypercube will have length

$$a = \left(\frac{V_{n-1}}{2^{dn}} \right)^{1/(n-1)} . \tag{13}$$

The computation of the coordinates of a hypercube center is made easier if the hypercubes are disposed on the simplex into layers that are parallel to the simplex basis and are numbered according to their distance from the reference vertex. The number of layers of a simplex with height H is given by

$$k^{max} = \lceil H/a \rceil . \tag{14}$$

For the standard simplex Δ_{n-1}, the i-th layer, identified by Δ^i_{n-2}, is the regular $(n-2)$-dimensional simplex at a distance $h^i_{n-1} = ia$ from the reference vertex. Figure 2 shows a 3-dimensional simplex and one of its layers.

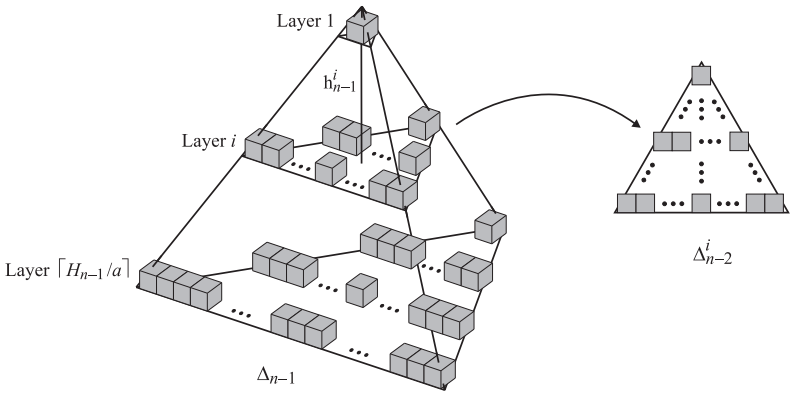

Figure 2: The standard $(n-1)$-dimensional simplex Δ_{n-1} and its i-th layer Δ^i_{n-2}.

The first thing that has to be done to locate an hypercube $y \in \{1, \ldots, 2^{dn}\}$ on Δ_{n-1} is to to find the index of its support layer, k_{n-1}. After that, it is necessary to find the position of the hypercube on the layer. This task may be accomplished dividing $\Delta^{k_{n-1}}_{n-2}$ into layers again. These layers are regular $(n-3)$-dimensional simplexes represented by Δ^j_{n-3}, where $j = 1, \ldots, \left\lceil H^{k_{n-1}}_{n-2}/a \right\rceil$ and $H^{k_{n-1}}_{n-2}$ is the height of $\Delta^{k_{n-1}}_{n-2}$. Each simplex Δ^j_{n-3} contains a set of $(n-3)$-dimensional hypercubes, obtained by projecting the original hypercubes onto this lower dimensional layer.

Applying recursively this strategy, we obtain $n-1$ layer indices that uniquely determine the coordinates of the center of a hypercube on Δ_{n-1}.

3.3. Computing the layer indices

Given a natural number y between 1 and 2^{dn} that represents one of the hypercubes in which the simplex was divided, we now show how to compute the $n-1$ layer indices used to calculate the coordinates of the center of the hypercube.

First, we note from (13) that the total number of hypercubes contained in Δ_{n-1} is V_{n-1}/a^{n-1}. Therefore, the number of hypercubes contained in the first k layers of the simplex is given by

$$\left(\frac{ka}{H_{n-1}}\right)^{n-1} \frac{V_{n-1}}{a^{n-1}} = \left(\frac{k}{H_{n-1}}\right)^{n-1} V_{n-1}. \tag{15}$$

The hypercube given by y will belong to layer k if

$$\left(\frac{k-1}{H_{n-1}}\right)^{n-1} V_{n-1} < y \leq k^{n-1} \left(\frac{k}{H_{n-1}}\right)^{n-1} V_{n-1},$$

or simply

$$k-1 < \sqrt[n-1]{\frac{y}{V_{n-1}}} H_{n-1} \leq k.$$

Therefore, the layer of Δ_{n-1} that contains hypercube y is given by

$$k_{n-1} = \left\lceil \sqrt[n-1]{\frac{y}{V_{n-1}}} H_{n-1} \right\rceil. \tag{16}$$

Once determined this layer index, we need to find the position of the hypercube y on the regular $(n-2)$-dimensional simplex $\Delta_{n-2}^{k_{n-1}}$. Naturally, the searched position depends on the content and the height of this simplex. However, these values also depend on

$$h_{n-1} = k_{n-1} a, \tag{17}$$

the distance between the layer and the reference vertex of $\Delta_{n-2}^{k_{n-1}}$.

Combining (8) and (9), we can define the content of $\Delta_{n-2}^{k_{n-1}}$ as

$$V_{n-2}^{k_{n-1}} = (c_{n-2})^{n-2} V_{n-2}, \tag{18}$$

where $c_{n-2} = k_{n-1}/k_{n-1}^{max}$ and k_{n-1}^{max} is given by (14), using $H = H_{n-1}$. In practice, since we have a huge number of hypercubes, we can use the approximate value

$$c_{n-2} \approx h_{n-1}/H_{n-1}. \tag{19}$$

From (10) and (12), we can also define the height of $\Delta_{n-2}^{k_{n-1}}$ as

$$H_{n-2}^{k_{n-1}} = c_{n-2} H_{n-2}. \tag{20}$$

Using (15) to define the number of hypercubes contained in the first $k_{n-1}-1$ layers of Δ_{n-1}, the index of the hypercube on $\Delta_{n-2}^{k_{n-1}}$ is given by

$$y_{n-2} = y - \left(\frac{k_{n-1}-1}{H_{n-1}}\right)^{n-1} V_{n-1}.$$

After determining this index, we may now work with an $(n-2)$-dimensional simplex. Applying the same procedure used to obtain (16), we define the layer of the hypercube y_{n-2} on $\Delta_{n-2}^{k_{n-1}}$ as

$$k_{n-2} = \left\lceil \sqrt[n-2]{\frac{y_{n-2}}{V_{n-2}^{k_{n-1}}}} H_{n-2}^{k_{n-1}} \right\rceil .$$

Thus, using (18) and (20), we get

$$k_{n-2} = \left\lceil \sqrt[n-2]{\frac{y_{n-2}}{V_{n-2}}} H_{n-2} \right\rceil .$$

To determine the remaining layers of the hypercube, we can apply

$$k_{n-i} = \left\lceil \sqrt[n-i]{\frac{y_{n-i}}{V_{n-i}}} H_{n-i} \right\rceil , \quad i = 3, \ldots, n-1,$$

where

$$y_{n-i} = y_{n-i+1} - \left(\frac{k_{n-i+1} - 1}{H_{n-i+1}} \right)^{n-i+1} V_{n-i+1}.$$

3.4. Determining the coordinates of the hypercube's center

Once we have the layer indices $k_1, \ldots, k_{n-1}$, let us show how to obtain $x = [x_1, x_2, \ldots, x_n]^T \in \Delta_{n-1}$, i.e., the vector that contains the Cartesian coordinates of the center of hypercube y. We will start using the first layer index k_{n-1} to compute the last coordinate x_n.

The distance between the reference vertex of Δ_{n-1} and the hyperplane that contains the bottom faces of the hypercubes on layer k_{n-1} is given by (17). Thus, the distance from the reference vertex to the hyperplane that passes through the center of the hypercubes on this layer is

$$\bar{h}_{n-1} = (k_{n-1} - 1/2)a.$$

Let us call $\bar{\Delta}_{n-2}^{k_{n-1}}$ the $(n-2)$-dimensional simplex that is at a distance $\bar{h}_{n-1}$ from the reference vertex of Δ_{n-1}. This simplex is parallel to $\Delta_{n-2}^{k_{n-1}}$, but passes through the center of the hypercubes of the k_{n-1} layer.

The n vertices of Δ_{n-1} (i.e. the points $e1, \ldots, e_n$) and the origin define a n-dimensional non-regular simplex S_n. Besides, a simplex that is similar to this one can be formed if we take e_n, the vertices of $\bar{\Delta}_{n-2}^{k_{n-1}}$, and the point $(0, \ldots, 0, x_n)$, as shown in Figure 3(a).

Due to the similarity between these n-dimensional simplexes, we obtain

$$x_n = 1 - \frac{\bar{h}_{n-1}}{H_{n-1}}.$$

After determining this last coordinate, we restrict our attention to the non-regular $(n-1)$-dimensional simplex S_{n-1} whose vertices are x_n and the vertices

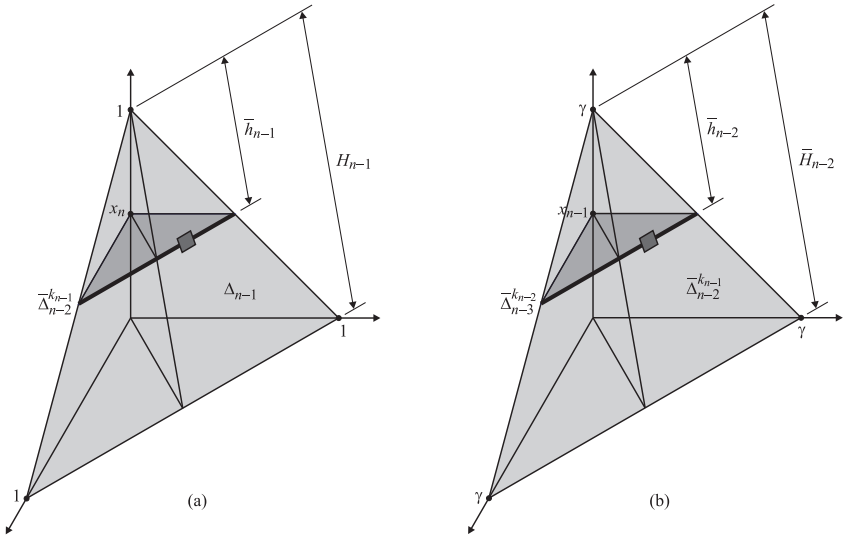

Figure 3: On the left, a schematic representation of an hypercube on Δ_{n-1} and the distances $\bar{h}_{n-1}$ and H_{n-1} used to compute the last coordinate of its center. On the right, a hypercube on $\bar{\Delta}_{n-2}^{k_{n-1}}$ and the distances $\bar{h}_{n-2}$ and $\bar{H}_{n-2}$ used to compute x_{n-1}.

of $\bar{\Delta}_{n-2}^{k_{n-1}}$. This simplex, marked in medium gray on Figure 3(a), belongs to a hyperplane that is parallel to the plane defined by $e_1, \ldots, e_{n-1}$. Therefore, we can use $\bar{S}_{n-1}$, the projection S_{n-1} onto this plane, to compute the remaining coordinates of x.

Besides the origin, the vertices of $\bar{S}_{n-1}$ are the points $\gamma e_1, \ldots, \gamma e_{n-1}$, where, $\gamma = \bar{h}_{n-1}/H_{n-1}$. These last $n-1$ points define the regular $(n-2)$-dimensional simplex $\bar{\Delta}_{n-2}^{k_{n-1}}$ on $\mathbb{R}^{n-1}$, as depicted in Figure 3(b).

The distance between the reference vertex of $\bar{\Delta}_{n-2}^{k_{n-1}}$ and the simplex $\bar{\Delta}_{n-3}^{k_{n-2}}$ that passes through the center of the hypercube is given by

$$\bar{h}_{n-2} = (k_{n-2} - 1/2)a.$$

Besides, from (19) and (20), we can define the height of $\bar{\Delta}_{n-2}^{k_{n-1}}$ as

$$\bar{H}_{n-2} = \frac{\bar{h}_{n-1}}{H_{n-1}} H_{n-2}.$$

Therefore, the $(n-1)$-th coordinate of x is given by

$$x_{n-1} = \gamma \left(1 - \frac{\bar{h}_{n-2}}{\bar{H}_{n-2}}\right) = \frac{\bar{h}_{n-1}}{H_{n-1}} - \frac{\bar{h}_{n-2}}{H_{n-2}}.$$

The remaining components of vector x are obtained repeating the procedure described above for the layer indices $k_{n-3}, \ldots, k_1$. Thus, we have

$$x_{n-i} \quad = \quad \frac{\bar{h}_{n-i}}{H_{n-i}} - \frac{\bar{h}_{n-i-1}}{H_{n-i-1}}, \quad i = 1, \ldots, n-2,$$

$$x_1 = \frac{\bar{h}_1}{H_1},$$

where $\bar{h}_j = (k_j - 1/2)a$.

3.5. Avoiding infeasible points

Depending on the number of hypercubes used to fill the simplex, it is possible that the center of a hypercube falls off the feasible region. This may happen because the number of hypercubes on a layer is computed using h_i and the coordinates of a hypercube center is computed using $\bar{h}_i$, distant $a/2$ from h_i.

This problem is only relevant if the number of hypercubes is really small. In this case, we suggest the use of hyper-cuboids in place of hypercubes. One possible way to define the edges of these hyper-cuboids is using the recursive relation

$$a_{n-i} = \frac{(k_{n-i+1} - 1/2)}{k_{n-i+1}} a_{n-i+1}. \tag{21}$$

where $a_{n-1} = a$. It is not hard to prove that the hypercube centers obtained using (21) belong to the feasible simplex.

4. A new genetic algorithm for simplex constrained combinatorial problems

In this section, we describe how to implement a genetic algorithm that uses the ideas described above to generate the whole efficient (or Pareto) frontier of the mixed integer portfolio optimization problem (4). It must be pointed out that this curve is not continuous, although being non decreasing.

One way to obtain points over the Pareto frontier is to choose L values of the penalty parameter λ, ranging from $\lambda = 0$ (which means that only the return is taken into account) to $\lambda = 1$, corresponding to a totally risk aversion investor. In spite of the fact that there are some criticism about this strategy, it is frequently used due to its simplicity.

The discretization of the solution space allow us to work directly with a binary vector to store the real part of the chromosome. Besides, an integer vector stores the indices of the assets included in the portfolio. With this chromosome representation scheme, we can use simple and efficient procedures for crossover and mutation.

A general scheme of the genetic algorithm is given by Algorithm 1 below. Details on the implementation are presented in the following subsections.

4.1. Initial population

We start from $\lambda = 0$, since the optimal solution is easy to obtain in this case. When $\lambda = 0$, only the asset returns are taken into account, and the best policy is to build up the portfolio with the n assets that have the highest expected return. Let the set I contain the indices of these assets and let k be the index of the most profitable asset. In this case, we define $x_k = 1 - \sum_{i \in I,\ i \neq k} l_i$, and the remaining assets receive $l_i, i \in I,\ i \neq k$.

Algorithm 1 General GA algorithm.

1: Generate an initial population.
2: Compute the fitness of the chromosomes.
3: **repeat**
4: Apply the crossover to a portion of the population.
5: Apply the mutation to some chromosomes.
6: Compute the fitness of the new chromosomes.
7: Select the chromosomes that will belong to the next generation.
8: **until** a stopping criterion is met.

Once this solution is available, we proceed solving the problems related to the remaining $L-1$ values of λ_j taking as the initial population the best solution obtained for λ_{j-1} along with a set of $n-1$ chromosomes with integer and real parts randomly generated.

4.2. Crossover

We use a standard one-point crossover on the real part and the OX crossover[2] on the integer part of the chromosome.

Each pair of parents generates two children. Besides, the chromosomes not selected for the crossover are duplicated, so the population is doubled in size at the end of the process.

Two parameters are used to control the crossover. The first, `CrossPerc`, defines the percentage of the population that undergo crossover. The second parameter, `CrossProb`, is the probability of applying the crossover to the real part of the chromosome.

4.3. Mutation

When applied to the real part of the chromosome, our mutation operator just flips one bit (change it from 0 to 1 or vice-versa). For the integer part, it is possible to swap two assets of the portfolio, or replace an asset by another that does not belong to the portfolio.

Three parameters are used to control the mutation: `MutPerc` is the percentage of the population that undergo mutation, `MutProb` is the probability of applying the mutation to the real part of the chromosome, and `MutProb2` is the probability of replacing an asset when the mutation occurs in the integer part of the chromosome.

4.4. Selection

A binary tournament scheme is used in the selection step. The last n chromosomes available after mutation are randomly reordered and paired with the first n chromosomes of the population. The best individual of each pair is selected.

[2]The OX crossover is a one-point crossover where only the non-repeated assets are swapped.

A very simple elitist scheme is also employed: at the end of an iteration, the worst chromosome of the population is replaced by the best individual found so far.

Notice that the chromosomes are always feasible, so no adjust is required for satisfying the cardinality or the bound constraints, as well as to return to the simplex.

5. Using the layer indices to represent the chromosome

The genetic algorithm presented above proved to be very effective for solving portfolio optimization problems. However, it requires the use of specific functions for dealing with very large integer numbers. Unfortunately, these functions may not be available in some environments, such as MATLAB. In this section, we show how this problem can be circumvented using the layers indices to represent the chromosomes.

As described in the previous sections, the generation of the vector of portfolio fractions from the real part of a chromosome can be done in two steps. Firstly, the hypercube number y is converted into the layer indices $k_1, \ldots, k_{m-1}$. Then, these indices are further converted into the coordinates of the hypercube center.

If dealing with large numbers is not allowed, we can suppress the first step of this scheme, and work directly with the layers indices. In this case, the integer part of the chromosome is kept unaltered, and the real part is stored in a binary vector b, that is divided into $n-1$ segments of equal length, $b^1, \ldots, b^{n-1}$, as shown below.

$$b = [\ \underbrace{0\ 1\ \ldots\ 1\ 1}_{b^1}\ \underbrace{1\ 0\ \ldots\ 1\ 0}_{b^2}\ \ldots\ \underbrace{1\ 1\ \ldots\ 1\ 1}_{b^{n-2}}\ \underbrace{0\ 1\ \ldots\ 0\ 0}_{b^{n-1}}\ ]$$

Each segment b^j is the binary representation of a natural number β^j that, on its turn, is used to compute the j-th layer index k_j. This index is given by

$$k_j = \left\lceil \frac{k_j^{max}}{k^{max}} \beta^j \right\rceil,$$

where $k_j^{max} = \lceil H_j/a \rceil$ is the largest number of layers in the j-th dimension and k^{max} is the number of bits required to store $\max_j\{k_j^{max}\}$ in binary format.

No change need to be done to Algorithm 1 in order to incorporate this new chromosome representation. However, since we use segments with equal lengths to store the indices of layers with different numbers of hypercubes, we may get a unbalanced distribution of points over the simplex. Fortunately, this problem may be avoided replacing our uniform random number generator by one that favors the layers with several hypercubes over the small layers. The new random number generator should be used to build the initial population as well as to select the points used in the crossover and mutation operators applied to the real part of the chromosomes.

The generation of an unbiased initial population, for example, is easy if we observe that the number of hypercubes contained in a layer k of an j-dimensional simplex in $\mathbb{R}^{j+1}$ is proportional to k^{j-1} (this result can be obtained combining (15) and simple formulas for the difference of powers). Therefore, we can assign to each natural number β^j, in which the vector b is decomposed, a random variable with the distribution function

$$F(\nu) = \begin{cases} \nu^j, & 0 \leq \nu \leq 1, \\ 0, & otherwise. \end{cases}$$

To obtain a random number with this distribution function, we generate a uniformly distributed random number $\mu \in [0, 1]$ and compute $\beta^j = F^{-1}(\mu) = \sqrt[j]{\mu}$. Once β^j is obtained, it is converted to a binary form and appended to vector b.

6. Numerical results

In this section, we compare the algorithms presented in Sections 4 and 5 to the algorithm proposed by Chang *et al.* [3]. The comparison is based on the five portfolio optimization problems introduced in [3], that use the stocks from the capital market indices Hang Seng (Hong Kong, 31 assets), DAX 100 (Germany, 85 assets), FTSE 100 (UK, 89 assets), S&P 100 (USA, 98 assets), and Nikkey 225 (Japan, 225 assets).[3]

The objective is not only to obtain the solution for a specific value of the risk or return, but to find the whole efficient frontier. Therefore, we define fifty equally-spaced values for the trade-off parameter $\lambda \in [0, 1]$, given in (4), and solve the corresponding problems. For all of the five problems, we require the portfolio to have exactly $n = 10$ assets. The lower limit for the investment on each asset is 0.01 (1% of the value invested on the whole portfolio).

The implementation of the genetic algorithm presented in Section 4 poses just one difficulty: it requires an efficient binary representation of a huge integer number. To cope with this problem, we decided to code the algorithm in Mathematica 7.0, since this software has many functions that apply binary operators directly to integer numbers of any size. As an alternative, it could be written in C or C++, using the GNU Multiple Precision Arithmetic Library (GMP)[4] to handle the real part of the chromosomes. On the other hand, the second version of the algorithm, presented in Section 5, is very easy to implement, and was built using MATLAB.

We define $d = 25$, so the chromosome representation of Section 4 requires less than 2/5 of the number of bits used by other genetic algorithms to store the real part of the population. The simplex is divided into about 10^{75} hypercubes

[3]The files containing these problems were downloaded from the URL http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/portinfo.html.

[4]The GMP library is available at http://gmplib.org.

with edges of length $a \approx 10^{-9}$. The number of layers of the whole simplex is $k^{max} \approx 10^9$. The population is formed by 100 chromosomes.

The new algorithm performs $13m$ iterations, so the number of crossovers and mutations are comparable to those presented by Chang *et al.*[5] [3]. After obtaining the portfolio for each value of λ, we keep the integer part of the chromosome and apply a quadratic programming algorithm to refine the real part. Other parameters used in the experiments are `CrossPerc` = 60%, `CrossProb` = 50%, `MutPerc` = 70%, `MutProb` = 50%, `MutProb2` = 10%. All of the tests were performed on a DELL Precision 5400 workstation, with a Intel Xeon E5430 processor (2,66 GHz, 4 CPUs), under the Windows Vista 64-bits operating system.

For all of the test sets, just a few assets of the optimal portfolio are changed when λ ranges from 0 to 0.6. Besides, although the amount of money invested on each asset may vary, most of them are kept at the lower level, suggesting that a better portfolio would be obtained relaxing the cardinality constraint. For these values of λ, all of the methods successfully found the optimal solution.

The main differences between the algorithms occur between $\lambda = 0.8$ and $\lambda = 1$. In this case, diversification is the key to obtain a good solution, so the percentage of investment is above the lower limit for most of the assets in the portfolio. For these high values of λ, the method of Chang et al. has some difficulty in keeping the solution inside the simplex, and is clearly outperformed by the new algorithm.

Tables 1 and 2 show the objective function values of the solutions found by the method of Chang *et al.* and the algorithm presented in Section 4, for the last ten values of λ. Results for the Hang Seng index are not shown since both algorithms arrived at almost the same solutions.

Table 1: Objective function values obtained by the method of Chang *et al.* and the algorithm presented in Section 4 for the DAX 100 and FTSE 100 problem sets.

λ	DAX 100		FTSE 100	
	New	Chang	New	Chang
0.816	-1.044E-03	-1.045E-03	-7.685E-04	-6.375E-04
0.837	-8.697E-04	-8.697E-04	-6.298E-04	-6.298E-04
0.857	-6.983E-04	-6.985E-04	-4.971E-04	-4.736E-04
0.878	-5.336E-04	-5.337E-04	-3.701E-04	-3.689E-04
0.898	-3.793E-04	-2.534E-04	-2.484E-04	-1.458E-04
0.918	-2.380E-04	-1.043E-04	-1.328E-04	-7.185E-05
0.939	-1.106E-04	4.895E-05	-2.364E-05	-1.947E-05
0.959	-1.529E-06	7.621E-05	6.767E-05	9.770E-05
0.980	8.770E-05	1.432E-04	1.440E-04	1.900E-04
1.000	1.482E-04	1.989E-04	2.060E-04	2.365E-04

[5]The algorithm of Chang *et al.*, performs one crossover and one mutation per iteration, and stops after $1000m$ iterations.

Table 2: Objective function values obtained by the method of Chang *et al.* and the algorithm presented in Section 4 for the S&P 100 and Nikkey 225 problem sets.

λ	S&P 100		Nikkey 225	
	New	Chang	New	Chang
0.816	-8.250E-04	-7,949E-04	-1.390E-04	-1.390E-04
0.837	-6.797E-04	-6.513E-04	-6.103E-05	-6.103E-05
0.857	-5.407E-04	-3.848E-04	1.335E-05	1.335E-05
0.878	-4.095E-04	-3.785E-04	8.161E-05	2.153E-04
0.898	-2.857E-04	-2.446E-04	1.420E-04	1.420E-04
0.918	-1.712E-04	-1.208E-04	1.950E-04	1.980E-04
0.939	-6.964E-05	-1.883E-05	2.404E-04	2.830E-04
0.959	1.984E-05	7.593E-05	2.737E-04	2.970E-04
0.980	8.553E-05	1.214E-04	2.935E-04	3.208E-04
1.000	1.345E-04	1.763E-04	3.048E-04	3.075E-04

A graphical comparison of the methods is presented in Figure 4, that shows the percentage change between the objective function values, Δf, given by the formula $\Delta f = 100(f_{Chang} - f_{New})/|f_{New}|$.

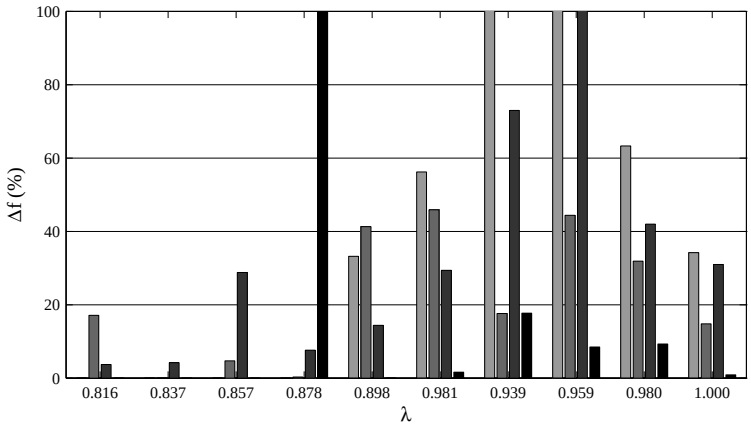

Figure 4: Percentage change between the objective function values obtained by the new algorithm and the method of Chang *et al.*.

We observe in Figure 4 that, for 17 problems, the objective function values obtained by the algorithm of Chang *et al* are at least 20% worse than those obtained by the new algoritm. Besides, for 11 problems, the difference is above 40%, and for 4 problems, it exceeds 100%.

Now, let us take a closer look at the the efficient frontiers of the five capital market indices. Clearly, these curves can be obtained not only from the best solutions found for the fifty values of λ, but also using some nondominated points

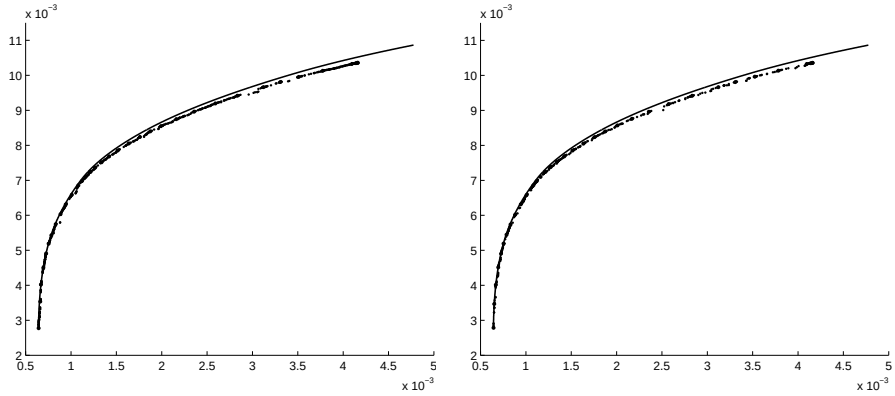

Figure 5: The efficient frontier for the Hang Seng problem. On the left, the solution obtained using the Chang *et al.* algorithm. On the right, the frontier generated by the new algorithm.

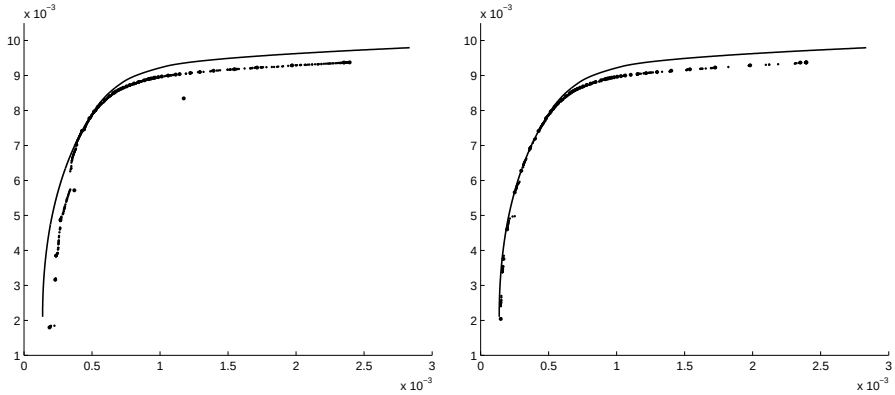

Figure 6: The efficient frontier for the DAX problem. On the left, the solution obtained using the Chang *et al.* algorithm. On the right, the frontier generated by the new algorithm.

found during the application of the genetic algorithms. Figures 5 to 9 present the efficient frontiers generated by both the algorithm of Chang *et al.* and the genetic algorithm introduced in Section 5. The dots represent the nondominated points obtained without using the quadratic programming local search strategy, while the continuous curve is the efficient frontier for the problem without the cardinality constraint, i.e. the curve obtained for problem (2).

Observing the left part of Figures 6 to 9, we note that the points obtained using the new algorithm are much closer to the unconstrained efficient frontier than the points generated by the algorithm of Chang *et al.*. This result is in accordance with Figure 4, since the left part of the efficient frontier corresponds to the higher values of λ. On the other hand, the upper right part of the frontier is almost the same for both methods, as expected. One may note that the distance between the frontier of the cardinality constrained problem and the

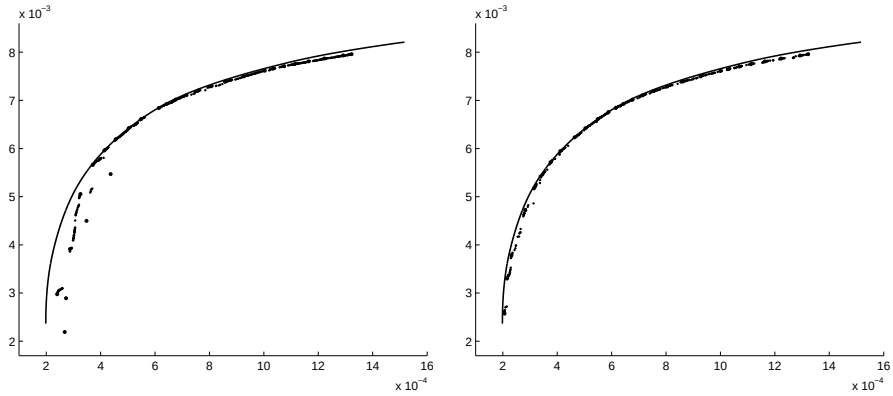

Figure 7: The efficient frontier for the FTSE problem. On the left, the solution obtained using the Chang *et al.* algorithm. On the right, the frontier generated by the new algorithm.

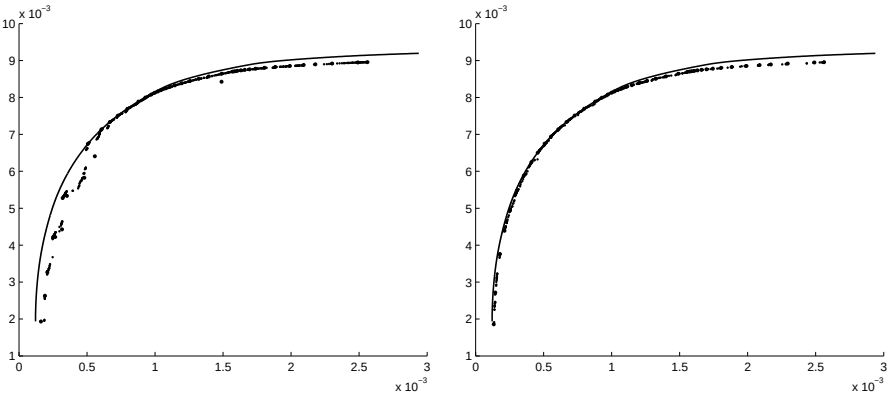

Figure 8: The efficient frontier for the S&P problem. On the left, the solution obtained using the Chang *et al.* algorithm. On the right, the frontier generated by the new algorithm.

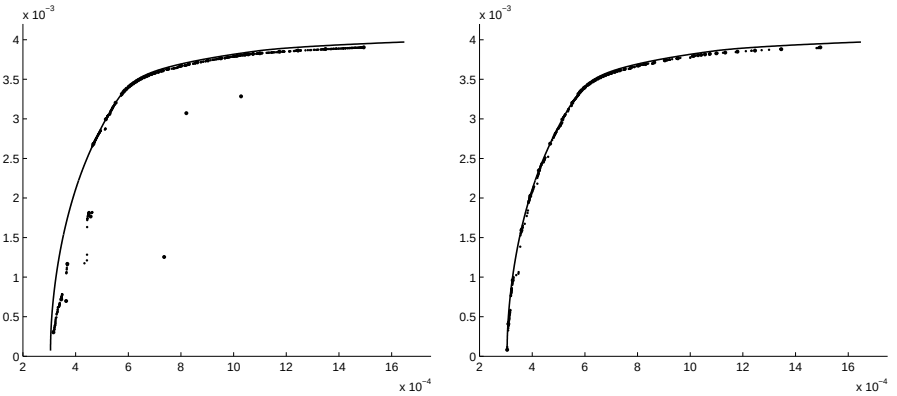

Figure 9: The efficient frontier for the Nikkey problem. On the left, the solution obtained using the Chang *et al.* algorithm. On the right, the frontier generated by the new algorithm.

curve of the uncostrained problem may be large for small values of λ, due to the existence of lower limits of investment for the assets in the portfolio.

7. Conclusions and future work

In this paper, we propose two new genetic algorithms for minimization over a simplex. The first algorithm requires the manipulation of very large integer numbers. The second algorithm uses a simpler scheme, so it may be coded in any programming language. The main feature of the algorithms is a new chromosome representation that approximates the simplex by a finite number of hypercubes.

The algorithms were applied to the solution of standard cardinality constrained portfolio problems, and shown a very good behavior when compared to the method of Chang *et al.* [3].

In the future, we plan to combine a very low resolution chromosome representation with the application of a local search strategy to a few individuals at the end of each iteration of the algorithm. We believe that this scheme will result in a efficient heuristic method for solving simplex constrained problems.

We also intend to compare our strategy to other chromosome representation schemes, such as the one proposed by Chiam *et al.* [4], and extend the approach to problems with more nonlinear objective functions.

References

[1] Bartholomew-Biggs, M.C. & Kane, S.J. A global optimization problem in portfolio selection, 2007. To appear in J. Comput. Manag. Science.

[2] Bertsimas, D. & Shioda, R. Algorithm for cardinality-constrained quadratic optimization, 2007. To appear in Comput. Optim. Appl.

[3] Chang, T.J.; Meade, N.; Beasley, J.E.; Sharaiha, Y.M. Heuristics for cardinality constrained portfolio optimization. Comp. Oper. Res., 2000, 27(13):1271-302.

[4] Chiam, S.C.; Tan, K.C.; Al Mamum, A. Evolutionary multi-objective portfolio optimization in practical context. Int. J. Autom. Comput., 2008, 5(1): 67-80.

[5] Devroye, L. Non-uniform random variate generation. New York: Springer, 1986.

[6] Di Gaspero, L.; di Tollo, G.; Roli, A.; Schaerf, A. Hybrid local search for constrained financial portfolio selection problems. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science, 4510. Berlin: Springer, 2007. p. 44-58.

[7] Fernandez, A. & Gomez, S. Portfolio selection using neural networks, Comp. Oper. Res., 2007, 34(4):1177-91.

[8] Jobst, N.J.; Horniman, M.D.; Lucas, C.A.; Mitra, G. Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints, Quant. Finance, 2001, 1(1):1-13.

[9] Le Thi, H.A. & Moeini, M. Portfolio selection under buy-in threshold constraints using DC programming and DCA. Proc. of Int. Conf. on Serv. Systems and Serv. Manag., 2006, p. 296 - 300.

[10] Maringer, D. & Kellerer, H. Optimization of cardinality constrained portfolios with a hybrid local search algorithm, 2003, OR Spectrum, 25: 481495.

[11] Markowitz, H. Potfolio Selection. Journal of Finance, 7: 77-91, 1952.

[12] Mitra, G.; Kyriakis, T.; Lucas, C.; Pirbhai, M. A review of portfolio planning: models and systems. In: Satchell, S.E., Scowcroft, A.E. (Eds.), Advances in portfolio construction and implementation. Oxford: Butterworth and Heinmann, 2003, p. 1-39.

[13] Moral-Escudero, R.; Ruiz-Torrubiano, R.; Suárez, A. Selection of optimal investment with cardinality constraints. In: Proceedings of the IEEE World Congress on Evolutionary Computation, Vancouver, Canada, 2006. p. 23822388,

[14] Skolpadungket, P.; Dahal, K.; Harnpornchai, N. Portfolio optimization using multi-objective genetic algorithms. Proc. IEEE Congr. on Evolut. Comput., Singapore, Malaysia, 2007, p. 516-23.

[15] Streichert, F.; Ulmer, H; Zell, A. Comparing Discrete and Continuous Genotypes on the Constrained Portfolio Selection Problem. In: Genetic and Evolutionary Computation, Lecture Notes in Computer Science, 3103. Berlin: Springer, 2004. p. 1239-50.

[16] Villela, P.F. An exact algorithm for portifolio optimization with cardinality constraints (in portuguese). MSc dissertation, Departamento de Matemática Aplicada, Universidade de Campinas, Campinas, Brazil, 2008.

[17] Weisstein, E.W. CRC concise encyclopedia of mathematics. 2.ed. Boca Raton: CRC, 2003.

[18] Xu, F.; Chen, W.; Yang, L. Improved particle swarm optimization for realistic portfolio selection. Proc. of 8th ACIS Int. Conf. on Software Eng., Artif. Intel., Networking, and Paral./Distr. Comput., 2007, p. 185-90.