Real-Polarized Genetic Algorithm for the Three-Dimensional Bin Packing Problem

André Homem Dornas
Computer Department
CEFET-MG
Belo Horizonte, MG, Brazil
andredornas299@gmail.com

Flávio Vinícius Cruzeiro Martins
Computer Department
CEFET-MG
Belo Horizonte, MG, Brazil
flaviocruzeiro@decom.cefetmg.br

João Fernando Machry Sarubbi
Computer Department
CEFET-MG
Belo Horizonte, MG, Brazil
joaosarubbi@gmail.com

Elizabeth Fialho Wanner
School of Engineering and Applied Science
Aston University
Birmingham, United Kingdom
e.wanner@aston.ac.uk

ABSTRACT

This article presents a non-deterministic approach to the Three-Dimensional Bin Packing Problem, using a genetic algorithm. To perform the packing, an algorithm was developed considering rotations, size constraints of objects and better utilization of previous free spaces (flexible width). Genetic operators have been implemented based on existing operators, but the highlight is the Real-Polarized crossover operator that produces new solutions with a certain disturbance near the best parent. The proposal presented here has been tested on instances already known in the literature and real instances. A visual comparison using boxplot was done and, in some situations, it was possible to say that the obtained results are statistically superior than the ones presented in the literature. In a given instance class, the presented Genetic Algorithm found solutions reaching up to 70% less bins.

CCS CONCEPTS

•Applied computing → Transportation; •Computing methodologies → Heuristic function construction; •Mathematics of computing → *Evolutionary algorithms;*

KEYWORDS

Bin Packing Problem, Genetic Algorithms, Genetic operators, Combinatorial optimization

ACM Reference format:
André Homem Dornas, Flávio Vinícius Cruzeiro Martins, João Fernando Machry Sarubbi, and Elizabeth Fialho Wanner. 2017. Real-Polarized Genetic Algorithm for the Three-Dimensional Bin Packing Problem. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071327

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071327

1 INTRODUCTION

In the Bin Packing Problem [10], given a set of objects with different dimensions and an unlimited amount of bins, all the items must be packed in order to minimize the number of bins used. This problem is part of the optimization class of Packing Problems [5, 6] and is commonly classified by: (i) the dimension of items, being 1D, 2D or 3D [18]; (ii) the bins size, that is, in case they are all with the same size, the problem is classified as homogeneous; otherwise, heterogeneous [3]; (iii) the constraints, such as weight limit, specific insertion order, overlapping and rotation possibility. In this paper, we studied the Three-Dimensional Bin Packing Problem (3D-BPP), a generalization of the Bin Packing Problem, with homogeneous containers.

The 3D-BPP is related to other similar problems: the Container Loading Problem [1, 9] and the Knapsack Loading Problem [12], differing mainly from the objective of each. In both, all objects must be packed in only one container. In the Container Loading Problem, the bin has an infinite length and the main goal is to achieve the maximum compressing yielding a minimization fo the bin length. In the Knapsack Loading Problem, the items have an associated value and the objective is to maximize the profit.

The 3D-BPP has great importance for many freight transport companies [17], such as mail and cargo ship companies, since a lot of money is spared by optimizing the space used and reducing the number of recipients used. In this way, improving those solutions benefit the use of these algorithms for frequent practical problems.

In this paper, a non-deterministic heuristic was proposed using a Genetic Algorithm (GA) with a Real-Polarized crossing operator to solve the homogeneous 3D-BPP. Also, a Local Search is executed in every generation, trying to generate a further improvement in the population. The objects to be loaded could be rotated in all directions and the only constraint considered was that one box could not be on top of another if it exceeds the dimensions (width and length) of the box below. The results of our algorithm (RPGA) were compared with the results presented in Zhu et al. [18] using a visual approach called boxplot. In 10% of instances, it was possible to say that the results of proposed proposed approach were statistically superior than the results in the literature. For a particular instance, we found about 70% less bins than BS-EPSD algorithm.

This work is organized as follows: Section 2 presents related works. Section 3 formalizes the 3D-BPP definition. Section 4 presents our proposed Genetic Algorithm. Section 5 presents the performed experiments and the results obtained. Section 6 concludes our work.

2 RELATED WORK

Several works related to the Bin Packing and the Container Loading Problems can be found in the literature. Understanding that these problems are very similar, some Container Loading works [2, 9, 15] were useful in this study. Referring to the Bin Packing Problem, we can cite the works [10, 18].

In Neto [15], the concept of subspace, used in this article, and the order of filling these spaces are defined. The author demonstrates the loading orders (top, side, front) and (side, top, front) and how to decompose the container space at each box. The author has implemented a genetic algorithm in which the objectives are to maximize the volume used, the weight, the stability and the monetary value of the load in a container. In addition, it demonstrates the fitness functions, calculated by a compound arithmetic mean, giving weights to each of the objectives.

George and Robinson [9] introduced the *flexible width* concept used on this article. The authors load the container by layers, so the *flexible width* is the capability to merge the present layer with the previous ones if there are space. We used this concept in this paper in the loading algorithm used in the GA decoding.

Cecilio and Morabito [2] proposed a change in the loading sequence previously proposed by George and Robinson [9]. The new sequence was determined by sorting the box according to the following criteria, considering, firstly, the most important: (i) Box with the largest dimension between the smallest ones (ii) Box with the largest quantity available (iii) Box with the largest dimension (iv) Box with the biggest volume (v) Box with the largest ratio: largest dimension by the smallest dimension We used this sequence in our Local Search strategy in Section 4 as CM_Sequence.

Zhu et al. [18] presented a non-deterministic algorithm to find a good solution to the Bin Packing Problem. This algorithm was designed using the *extreme point insertion heuristic* [4] with two improvements based on *Space Defragmentation*, which are: the *push-out* operation, consisting of, just before inserting a box, all items that intercept it are pushed forward to make the box fit into space; the *inflate* operation, which inflates a box, already inserted, as much as possible, pushing the other boxes already placed, and then verifies if the box to be inserted fits in the inflated space, swapping it by the inflated box. The authors used the 3D-BPP instances generated by Martello et al. [11] and new instances created by themselves to compare their algorithm with several already existing in the literature known to produce good results. Thus, the proposal presented here will be compared with the results displayed by Zhu et al. [18].

In Takahashi et al. [16] the authors proposed a new operator for continuous GAs called Real-Polarized Crossing. One of the characteristics of this operator is to generate children that are more likely to be close to parents of better fitness value. In a concept of geometric operators, Martins et al. [13] presented a version of the Real-Polarized Crossing operator for discrete problems.

5	1	3	2	4	6	8	10	9	7
0	1	2	3	4	5	6	7	8	9

Figure 1: An individual representing a solution in which the first box loaded is the 5, then the 1, then the 3, and so on.

Our work differs from others since we used the Real-Polarized operator for the crossover in the GA, producing results that surpassed others in the literature. Also, our loading algorithm merged the best characteristics of others loading algorithms [2, 9, 15].

3 PROBLEM DEFINITION

Given a set of n rectangular-shaped items $I = \{1, ..., n\}$, defined by a width w_i, a height h_i and a length l_i ($i \in I$), and a set C of infinite number of identical bins with width W, height H and length L, the Three-Dimensional Bin Packing Problem consists in load all n items orthogonally, inside a certain number of bins and minimizing the total number of bins used.

Many constraints can be added to the 3D-BPP to specify the problem. It can be considered rotations in all directions and, in some situation, some orientations can be restricted. The bins size can be equal or not. The boxes can have weights and each container a weight limit (if homogeneous, all bins have the same limit). It can be considered an insertion order, due to an relation with the unload order [8].

In this study, we consider that the length L of the bin is parallel to the Z-axis of the Cartesian coordinate system; the width W is parallel to the X-axis; and the height H is parallel to the Y-axis. All bins are considered to be in the first octant and one corner is at origin. We assumed that the boxes could be rotated in all six possibilities.

4 REAL-POLARIZED GENETIC ALGORITHM FOR THREE-DIMENSIONAL BIN PACKING PROBLEM

A Genetic Algorithm (GA) [14] is a metaheuristic based on the natural processes of evolution, considering the crossing of individuals, the gene mutation process and natural selection, that is, an individual who is better adapted (better genetic conditions) will have greater chances of surviving and passing on their genes to the next generations. The algorithm works using the idea that each iteration is a generation of the population, thus, all the processes described above are used, in order to keep the population size constant and to generate diversified individuals (problem solvers), each time better.

The proposed Real-Polarized Genetic Algorithm for Bin Packing (RPGA) was based on the GA proposed by Neto [15], in which the loading pattern is based on the upper, lateral and frontal order, to maximize the volume used of the container. Also, in every generation, the best individual is submitted to a Local Search.

4.1 Coding

An individual, or chromosome, represents a solution of the algorithm. Each individual is a list of boxes representing a loading sequence of the instance, that is, each allele represents a single box. Figure 1 shows a chromosome sample.

4.2 Decoding

For the GA construction proposed here, a deterministic algorithm was implemented to insert boxes into a container and evaluate an individual. We decided to implement a new loading algorithm instead of using already existing ones, such as George and Robinson [9], due to some particularities of the instances. During the loading process, the subspace concept is used, which represents a lateral, superior or frontal space with respect to the container, which has a size and spatial coordinates.

The general idea of the constructed loading algorithm is represented in Algorithm 1. Basically, we have two distinct procedures: (i) Try_Load_Box, which returns if a box can be inserted inside a certain bin; (ii) Load_Box, which does the loading itself, generating the necessaries subspaces. The latter procedure is explained next.

Algorithm 1: Loading

```
Data: C, I
foreach c in C do
    foreach i in I do
        if not Try_Load_Box(i, c) then
            c ← c + 1 ;
        end
        if c is ∅ then
            Load_First_Box(i, c) ;
        else
            Loading_Subsequent_Boxes(i, c) ;
        end
    end
end
```

4.2.1 Try_Load_Box Procedure. This procedure makes a verification if it is possible to insert a box in the current bin. In this way, if it is not possible to insert in the current bin, a new empty one is generated and the loading continues inserting the next box as the first box.

4.2.2 Load_First_Box Procedure. The first inserted box in the container is placed in the position (0,0,0), that is, touching the floor, the back and the left wall. From this box, the first subspaces are created, being: the superior, the subspace that has the same width and depth of the box and height equal to the top of the container until the top of the box; the side, which has the same depth as the box, height equal to the height of the container and width relative to the lateral space of the carton; the front, which has the same height and width of the container and depth equal to the free space in front of the container.

4.2.3 Loading_Subsequent_Boxes Procedure. The order of insertion attempt was first in the upper subspace, then in the lateral and finally in the frontal, as shown by Neto [15], since it is understood that this is the order commonly used for a real application of the algorithm. When inserting a box, the first step is to check if is possible to merge subspaces, then we try to insert it making rotations every time it does not fit, only changing of subspace after all rotations. These steps are described as follows:

Use of Previous Side Subspaces. Based on the *flexible width* concept [9], when a box is inserted into the side subspace, a new subspace (S1) is created, updating the width. If the next box does not fit into this new subspace, it will be inserted in the front part, creating a new subspace (S2). The Figure 2a shows this configuration with S1 and S2. Thus, to take advantage of space S1 left behind, if S2 has a x coordinate greater than or equal to the x coordinate of S1, spaces S1 and S2 are grouped, so that their depth is summed and the new subspace has a coordinate x correspondent to the greater x between the two, as shown in Figure 2b. Then, the next box is inserted in the new subspace (S1+S2), resulting in the configuration shown on Figure 2c.

Rotation. A box can be rotated to fit in a space and this rotation was done by changing the value of two dimensions at a time, for 6 times, intercalating the following changes: width with depth and height with width. In this way, it is possible to rotate the object in all six possibilities. The Figure 3 shows all the possibles orientations of a box considering the axes.

4.3 Initial Population

In this Section, we present the proposed strategy to generate the initial population. The initial population P consists of *popSize* individuals in which each one of these is generated using the Restricted Candidate List (RCL) concept, utilized in the Construction Phase of the GRASP (Greedy Randomized Adaptive Search Procedure) algorithm [7]. The Algorithm 2 shows the general process of creating the initial population.

At the Algorithm, a randomized greedy technique provides feasible solutions. Each feasible solution is iteratively constructed, one element at a time. However, instead of always selecting the best solution, a Restricted Candidate List (RCL) of good elements is built, and one element (not necessarily the top candidate) is randomly selected. Algorithm 3 presents the proposed Generate_Individual algorithm.

We use, as a Build_RCL Procedure, a algorithm to order the items according to their volume. However, instead of choosing the item with maximum volume, we create a list of items and randomly select one element of the list. An RCL parameter α that can vary from 0.0 to 1.0, determines the level of greediness or randomness at the Generate_Individual. When $\alpha = 0.0$, the Generate_Individual becomes a simple greedy algorithm and will always select the item with maximum volume at each iteration. Otherwise, when $\alpha = 1.0$, the Generate_Individual becomes totally random. If the RCL is built with many elements, then many different solutions will be produced, according to chosen α value. The Algorithm 4 presents a pseudo-code from the Build_RCL Procedure.

The Build_RCL Procedure works as follows: first, the RCL list is set to empty. Then we compute the Maximal and Minimal volume of all items. For each remaining item, we verify if the volume of this item is more than $Max - \alpha(Max - Min)$. If the condition is true, we add this item to the RCL list.

4.4 Evaluating the Population

Each individual of the population is loaded using the sequence suggested by Neto [15], as quoted in Section 4.2. Then, we evaluate the solution using as the first criterion the number of containers

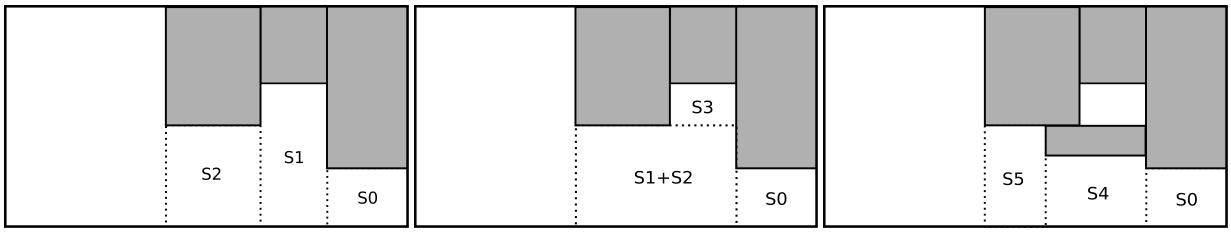

Figure 2: Example of joining previous subspaces while loading the container. In Figure 2a, subspaces S2 and S1 can be joined. Figure 2b shows the subspace resulting from the union of S1 and S2, that can contain a box with greater depth than with S1 and S2 separated. Figure 2c shows the result of inserting a box after the union.

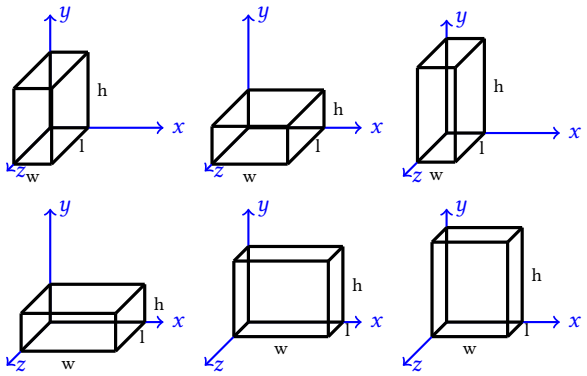

Figure 3: All the possibles rotations of an object in the 3 axes

Algorithm 2: Population Generation

```
Data: popSize, I, α
P ← ∅;
while size(P) < popSize do
    P ← P + Generate_Individual(I, α);
end
return P;
```

Algorithm 3: Generate_Individual Procedure

```
Data: I, α
I' ← Sort_Volume(I); /* Sort by volume decreasingly */
Individual ← ∅;
while I' is notempty do
    RCL ← Build_RCL(I',α); /* Create RCL */
    Selected_Item ← Randomly_Choose_Element(RCL);
    Individual ← Individual + Selected_Item;
end
return (Individual);
```

used. We consider that the best solutions are the ones with the least amount of containers. When there is a tie in the number of containers, we use as the criterion the lowest sum of void volume with inverse quadratic relation, that is, a large empty volume in

Algorithm 4: Build_RCL Procedure

```
Data: I', α
Max ← Select_Max_Volume(I'); /* Max volume */
Min ← Select_Min_Volume(I'); /* Min volume */
RCL ← ∅ ;
/* Building RCL */
foreach i in I' do
    if Volume(i) >= Min + α * (Max − Min) then
        RCL ← RCL + i;
    end
end
return (RCL);
```

the first container is worse than in the second one (Equation 1). In this way, solutions in which the first containers are fuller are better evaluated.

$$\sum_{j=1}^{maxBins} \frac{(totalVolume - \sum boxVolume)}{totalVolume} * \frac{1}{j^2} \quad (1)$$

4.5 Genetic Operators

Genetic operators are the elements responsible for varying individuals in the population, in order to find better solutions each time, maintaining the adaptive characteristics acquired in previous populations and, at the same time, diversifying solutions. The most commonly used operators are mutation, crossover and selection of individuals.

In this work, the operators implemented in the Genetic Algorithm were based on operators already known in the literature. They can be considered generic because they serve for any genetic algorithm that uses an integer representation of the solution. However, the crossover operator is more elaborate since it breeds the new individuals so that they look more like the better parent.

4.5.1 Mutation. A mutation operator was designed in order to create a disturbance in the population, seeking to generate solutions not yet verified. This operator bases on the position change of a defined alleles number in the chromosome. The number of boxes to be exchanged was determined in as *tExch* of the total items of

1	1	5	2	2	2	4	4	3	5
0	1	2	3	4	5	6	7	8	9

(a) A solution representation based on the type of each box, that is, boxes with same dimensions have the same value. If the box in position 3 is drawn, then it will be swapped with the box in position 6, the next one which is not of type 2

1	1	5	4	2	2	2	4	3	5
0	1	2	3	4	5	6	7	8	9

(b) Solution representation of (a) after swapping of the boxes. The mutation consists of repeating this process several times ($tExch$ of total boxes) for the same individual

Figure 4: Mutation operation example

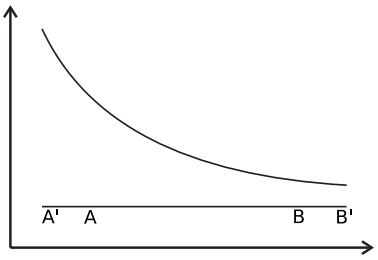

Figure 5: Extrapolation of parents A and B

the individual and the probability of a mutation occur in a subject was set as $tMut$.

Taking into account the existence of same box types (with the same dimensions), the mutation, illustrated in Figure 4, consists of selecting a random box in the individual and switching with the first box, next to it in the list, that is not of the same type. In this way, it is avoided to exchange boxes of the same kind, which would not make any difference in the solution.

4.5.2 Crossover. The crossover operator used is based on the Real-Polarized Crossing operator and uses the concept of *edit move*, as shown by Martins et al. [13], an artifice used to construct geometric operators. The minimum path between two individuals is the minimum set of *edit moves* required for the two individuals to be equivalent. Also, the concept of distance between two individuals was used, which is equivalent to this minimum path length between them.

To generate more differentiated solutions, not only through the mutation, the distance between two chromosomes was extrapolated, that is, the distance between them is increased by 10% for each side, as shown in Equation 2. The generation of the children is performed after the extrapolation, using the parents A' and B'.

$$\begin{aligned} dist(A', A) = dist(B', B) = 0.1 * dist(A, B) \\ dist(A', B') = 1.2 * dist(A, B) \end{aligned} \quad (2)$$

The children are created from *edit moves* starting from the best-evaluated parent, that is, the children are in the path between A' and B'. In order to maintain the best individuals, the amount of *edit move* to be performed for the first child is determined randomly according to a quadratic distribution so that the number of exchanges is more likely to generate an individual closer to the best parent, whereas for the second child a uniform distribution is used. Figure 5 demonstrates this quadratic distribution across the curve above the parent extrapolation line, considering that the best parent, in this case, is A. Parents are chosen randomly, where everyone has the same chance of being chosen. The probability of a crossover occur in one generation is $tCross$.

4.5.3 Selection. The population size is raised after the crossover operation, so a tournament selection method is used to remove one individual at a time, seeking to maintain the population size constant. This selection function, illustrated in Algorithm 5, chooses two individuals randomly and the worst of them is selected to be removed from the population. This method is repeated until the population returns to its original size.

Algorithm 5: Select Survivors

```
Data: P, popSize
while size(P) > popSize do
    c1 ← Random(P); /* choose one individual randomly */
    c2 ← Random(P); /* choose one individual randomly */
    P ← P − Worst(c1, c2);
end
return P;
```

4.6 Local Search

Seeking always to find better solutions, at each iteration, a local search on the best individual *best* is performed. This search was implemented using the loading sequence defined by Cecilio and Morabito [2].

This local search method was implemented using the Refining Heuristic known as the First Improvement Method. Therefore, in each iteration, the Algorithm 6 procedure is done, and if a better solution than the original is found, it is returned and added to the population, ending the process. However, if no best solution is found, reaching the iteration threshold set at $localIter$, the process does not return any new solutions.

Algorithm 6: Local Search

```
Data: Individual
B ← ∅ ;
Q ← Position(Random_Box(Individual));
B ← Remove_Items_After_Position(Individual, Q − 1);
while B is not ∅ do
    A ← Choose_Item_From(B); /* Choose item based on
       CM_Sequence */
    Individual ← Individual + A ;
end
return Individual;
```

5 EXPERIMENTS

To evaluate the efficiency of the RPGA, experiments were performed on two sets of instances suggested by Zhu et al. [18]. The tests were

Table 1: Comparison between the algorithms for the 3D-BPP instances with respect to the best solution in the 11 executions

Algorithm	Wins	Loses	Percentage
BS-EPSD	235	85	73,4%
RPGA	85	235	26,6%

Table 2: Comparison between algorithms for 3D-BPP instances through *boxplot*

Algorithm	Wins	Loses	Undefined	Percentage
RPGA	20	282	18	6,25%
BS-EPSD	282	20	18	88,13%

performed on a computer with AMD FX-8350 4GHz with 16GB of RAM, running Linux Mint 17.3 Cinnamon 64-bit. The algorithm was implemented using Oracle's 64-bit Java Development Kit. In all tests, $popSize = 100$, $tMut = 5\%$, $tExch = 5\%$, $tCross = 85\%$, $localIter = 20$, and, $\alpha = 0.75$. All this parameters were defined empirically.

The first set of instances generated by Martello et al. [11], called 3D-BPP, contains 320 instances. This set is divided into 8 classes, each class divided into 4 groups and each group has a given number of objects: 50, 100, 150 and 200. For each of these instances, the algorithms to be compared were executed 11 times.

Table 1 shows a comparison of the best solution of the 11 executions of each algorithm for each instance. An algorithm solution was considered better when it presented a smaller number of containers. Comparing the number of containers in the 320 instances is possible to see that in 26.6% of the instances the RPGA performed better than the BS-EPSD.

For a more consistent analysis, *boxplot* type graphics were plotted for the 320 instances for both algorithms. In this way, they were compared side by side to visually check if the "boxes" of the *boxplot* does not overlap, and identify which "box" is further down, thereby identifying which approach was considered best. Then, with this visual analysis it was possible to determine 3 situations:

i Confirm when RPGA solution was statistically better than BS-EPSD solution. (Figure 6a)
ii Not to conclude which approach was statistically better, since it would require another statistical test for such a conclusion. (Figure 6b)
iii Confirm when BS-EPSD solution was statistically better than RPGA solution. (Figure 6c)

A general analysis can be done by looking at the graph in Figure 7. The circles (in red) and the x represent, respectively the BS-EPSD and RPGA solutions absolute values, in two colors: blue for when RPGA performed worse than BS-EPSD and black, otherwise. It can be noticed that the RPGA obtained better results for the smaller instances. However, even so, the results for the class shown in Figure 7 approximated the BS-EPSD results. This class was the one with the best results in the comparison.

Table 3: Comparison between algorithms for 3D-BPP instances through *boxplot* per class

Algorithm	RPGA	BS-EPSD	Undefined
Class 1	0	38	2
Class 2	2	37	1
Class 3	0	39	1
Class 4	11	21	8
Class 5	4	35	1
Class 6	0	40	0
Class 7	2	36	2
Class 8	0	37	3

Table 2 shows the comparison between RPGA and BS-EPSD in all 3D-BPP instances, while Table 3 shows the same data divided into classes, confirming that the best RPGA performance was in Class 4. In this way, Figure 8 illustrates the percentage difference between RPGA and BS-EPSD (displayed by the black line) in the Class 4 instances, demonstrating that, when the RPGA got better results, it achieved a bigger difference that when it got worse results. In average (the mean of the sum of the gains minus the losses), considering all this class, the RPGA got 5% less bins than BS-EPSD, reaching up to about 70% fewer bins. When the performed worst in this class, the biggest difference was at most 20% more bins, while there were at least 8 instances that achieved a gain of more than 20%.

Table 4: Comparison between the algorithms for the T3 instances with respect to the best solution in the 11 executions

Algorithm	Wins	Loses	Percentage
BS-EPSD	81	19	82%
RPGA	19	81	18%

The second set of instances generated by Zhu et al. [18], called *Type3* (T3), has 100 instances and was built based on a real problem data for loading boxes into a 20-foot container. This set of instances is separated into two classes and each of them is divided into 5 groups of 2000, 4000, 6000, 8000 and 10000 items containing 10 instances each. For each of the instances, the algorithm was also executed 11 times. Table 4 shows a comparison between the best solution of the 11 executions of each algorithm for each instance. Again an algorithm was considered better when it presented a smaller number of containers.

The same analysis made for the 3D-BPP instances using the *boxplots* also was used for the T3, in order to make a further results evaluation. The results obtained are shown in Table 5 and in Figure 9. Also, the Table 6 shows the comparison between the algorithm divided by the two T3 classes.

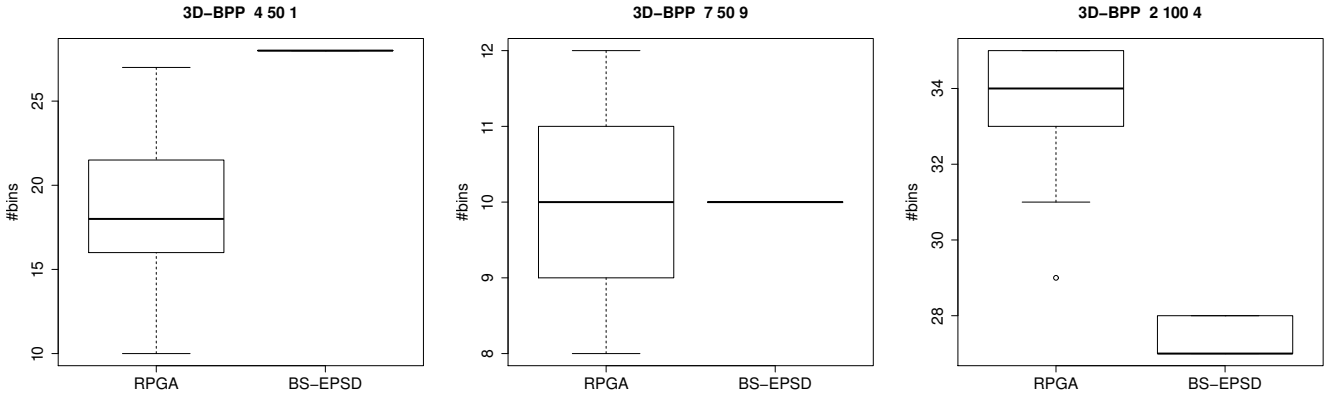

Figure 6: This Figure shows three *boxplots* comparing the 3D-BPP instances solutions of RPGA and BS-EPSD algorithms. Figure 6a shows an example which the RPGA solution was statistically better than the BS-EPSD solution. Figure 6b shows an example which it is not possible to statistically determine a better solution. Figure 6c shows example which the BS-EPSD solution was statistically better than the RPGA solution

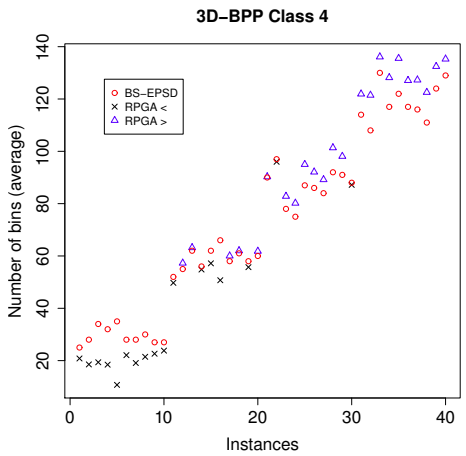

Figure 7: An example of the result of the Class 4 instance. All instances of 3D-BPP in relation to the average number of containers found. Each of the 8 classes occupies 40 indexes and each size (in ascending order) within them, occupies 10 indexes.

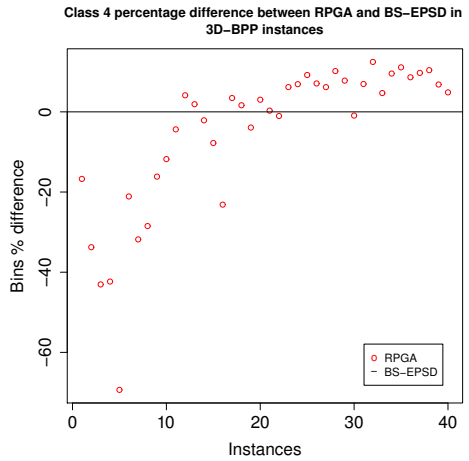

Figure 8: The results obtained for Class 4 of 3D-BPP instances for both algorithms, fixing BS-EPSD and comparing which bins percentage the RPGA achieved. A negative percentage means that RPGA performed better, that is, a difference of -20 means that RPGA achieved 20% less bins than BS-EPSD. A positive percentage means that RPGA performed worse, getting solutions with more containers than BS-EPSD.

Table 5: Comparison between algorithms for T3 instances using *boxplot*

Algorithm	Wins	Loses	Undefined	Percentage
BS-EPSD	81	17	2	81%
RPGA	17	81	2	17%

Analyzing the two graphs of Figure 4, it can be seen that the results of the RPGA closely approximated the results of the BS-EPSD and, in some cases, the RPGA obtained a much better result than the literature, especially for large instances.

Table 6: Comparison between algorithms for T3 instances through *boxplot* per class

Algorithm	RPGA	BS-EPSD	Undefined
Class 1	7	43	0
Class 2	10	38	2

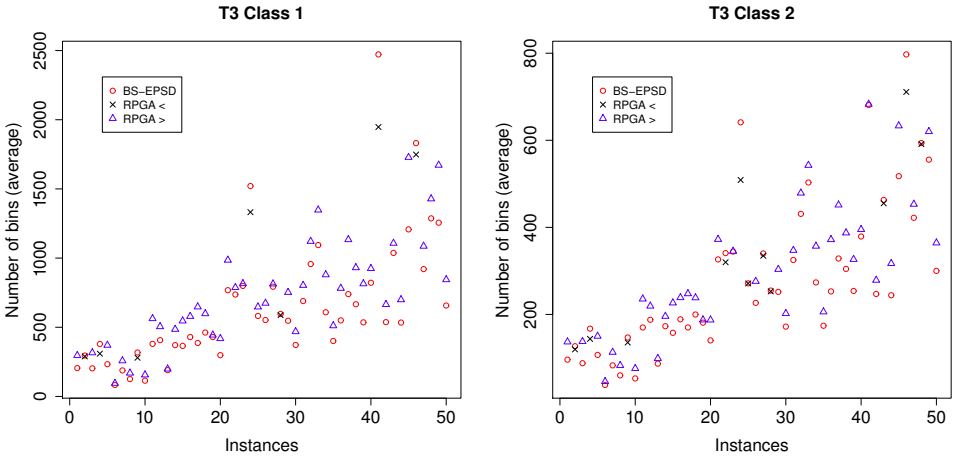

Figure 9: Comparison between the algorithms for the two T3 instance classes

In both tests instances, it was possible to notice that in some cases the RPGA performed better than BS-EPSD. In general, RPGA did better for smaller instances, although it did achieve some good results for large instances. This may have occurred not only because of the number of items to be packaged but also because of the format of the items, which may favor a better filling for the RPGA rather than BS-EPSD.

6 FINAL CONSIDERATIONS

From the presented results, it can be concluded that the algorithm implemented produces satisfactory solutions for practical applications. This can be confirmed since the algorithm used for comparison, BS-EPSD, was compared with several others in the literature as shown by Zhu et al. [18], having equal or better results than those already existing. In addition, the T3 instances produced by the BS-EPSD author are instances of a real problem, where the data used were made available by a logistics company, and the results of the tests using T3 showed that the RPGA managed to find solutions better than those found by Zhu et al. [18].

It was possible to notice that the RPGA was better in instances with smaller quantities of objects by checking the results obtained. As future works, the instances will be studied better in order to adapt the RPGA to make it more general so that its performance can be also efficient in large instances. This can probably be achieved by improving the utilization of unused spaces in the loading algorithm and also by modifying and testing new sequences to load the objects in the local search.

In future works, new genetic operators will be implemented for the algorithm based on geometric operators. These new operators would be made specifically for the Bin Packing Problem so that it is possible to use unique aspects of this category of problems in order to produce better and better solutions.

Acknowledgements

This work was partially funded by CNPq, FAPEMIG and CAPES.

REFERENCES

[1] Eberhard E Bischoff, F Janetz, and MSW Ratcliff. 1995. Loading pallets with non-identical items. *European journal of operational research* 84, 3 (1995), 681–692.
[2] FO Cecilio and Reinaldo Morabito. 2004. Refinamentos na heurística de George e Robinson para o problema de carregamento de caixas dentro de contêineres. *Transportes* 11, 2 (2004), 32–45.
[3] CS Chen, Shen-Ming Lee, and QS Shen. 1995. An analytical model for the container loading problem. *European Journal of Operational Research* 80, 1 (1995), 68–76.
[4] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. 2008. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing* 20, 3 (2008), 368–384.
[5] Kathryn A Dowsland and William B Dowsland. 1992. Packing problems. *European journal of operational research* 56, 1 (1992), 2–14.
[6] Harald Dyckhoff. 1990. A typology of cutting and packing problems. *European Journal of Operational Research* 44, 2 (1990), 145–159.
[7] Thomas A. Feo and Mauricio G. C. Resende. 1995. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization* 6, 2 (1995), 109–133. DOI: http://dx.doi.org/10.1007/BF01096763
[8] Michel Gendreau, Manuel Iori, Gilbert Laporte, and Silvano Martello. 2006. A tabu search algorithm for a routing and container loading problem. *Transportation Science* 40, 3 (2006), 342–350.
[9] John A George and David F Robinson. 1980. A heuristic for packing boxes into a container. *Computers & Operations Research* 7, 3 (1980), 147–156.
[10] Silvano Martello, David Pisinger, and Daniele Vigo. 2000. The three-dimensional bin packing problem. *Operations Research* 48, 2 (2000), 256–267.
[11] Silvano Martello, David Pisinger, Daniele Vigo, Edgar Den Boef, and Jan Korst. 2007. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software (TOMS)* 33, 1 (2007), 7.
[12] Silvano Martello and Paolo Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.
[13] F. V. C. Martins, E. G. Carrano, E. F. Wanner, R. H. C. Takahashi, G. R. Mateus, and F. G. Nakamura. 2014. On a Vector Space Representation in Genetic Algorithms for Sensor Scheduling in Wireless Sensor Networks. *Evolutionary Computation* 22, 3 (2014), 361 – 403. DOI: http://dx.doi.org/10.1162/EVCO_a_00112
[14] Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.
[15] Lúcio Lopes Rodrigues Neto. 2005. *Um Algoritmo Genético para Solução do Problema de Carregamento de Ccontainer*. Ph.D. Dissertation. Universidade Federal do Rio de Janeiro.
[16] R. H. C. Takahashi, J. A. Vasconcelos, J. A. Ramirez, and L. Krahenbuhl. 2003. A multiobjective methodology for evaluating genetic operators. *IEEE Transactions on Magnetics* 3, 39 (2003), 1321–1324.
[17] Tian Tian, Wenbin Zhu, Andrew Lim, and Lijun Wei. 2016. The multiple container loading problem with preference. *European Journal of Operational Research* 248, 1 (2016), 84–94.
[18] Wenbin Zhu, Zhaoyi Zhang, Wee-Chong Oon, and Andrew Lim. 2012. Space defragmentation for packing problems. *European Journal of Operational Research* 222, 3 (2012), 452 – 463. DOI: http://dx.doi.org/10.1016/j.ejor.2012.05.031