A PARALLEL MULTI-POPULATION BIASED RANDOM-KEY GENETIC ALGORITHM FOR A CONTAINER LOADING PROBLEM

JOSÉ FERNANDO GONÇALVES AND MAURICIO G. C. RESENDE

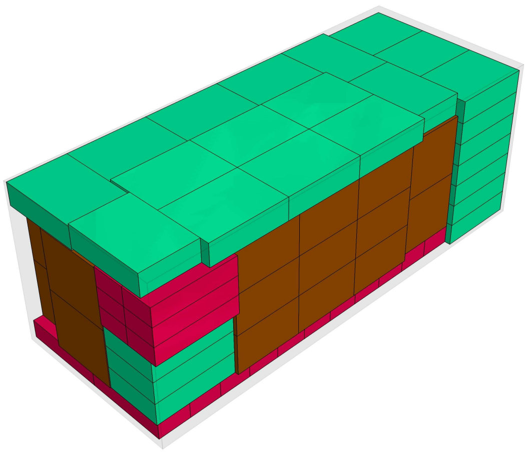
Abstract. This paper presents a multi-population biased random-key genetic algorithm (BRKGA) for the Single Container Loading Problem (CLP) where several rectangular boxes of different sizes are loaded into a single rectangular container. The approach uses a maximal-space representation to manage the free spaces in the container. The proposed algorithm hybridizes a novel placement procedure with a multi-population genetic algorithm based on random keys. The BRKGA is used to evolve the order in which the box types are loaded into the container and the corresponding type of layer used in the placement procedure. A heuristic is used to determine the maximal space where each box is placed. A novel procedure is developed for joining free spaces in the case where full support from below is required. The approach is extensively tested on the complete set of test problem instances of Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) and is compared with other approaches. The test set consists of weakly to strongly heterogeneous instances. The experimental results validate the high quality of the solutions as well as the effectiveness of the proposed heuristic.

1. Introduction

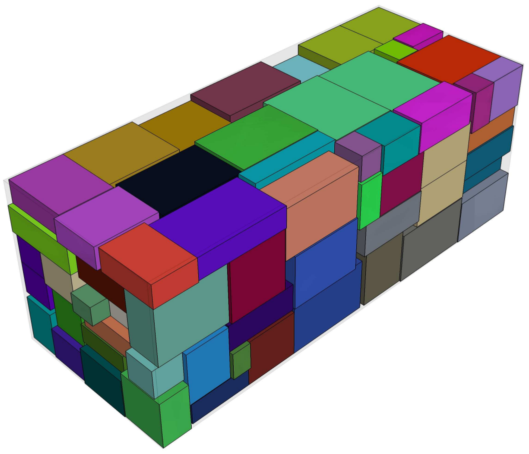
The *Single Container Loading Problem* (CLP) is a three-dimensional packing problem in which a large rectangular box (the container) has to be filled with smaller rectangular boxes of different sizes. Figure 1.1 shows that CLPs can be differentiated according to the mix of box types to be loaded. They vary from the *completely homogeneous* case, where boxes have identical dimensions and orientations, to the *strongly heterogeneous* case, where boxes of many different sizes are present. CLPs with relatively few box types are often referred to as *weakly heterogeneous* (Bischoff and Ratcliff, 1995). According to the typology of Wäscher et al. (2007) for cutting and packing problems, the heuristic for the CLP presented in this paper falls into the output maximization assignment category and can be applied to both the *Single Large Object Placement Problem* (3D rectangular *SLOPP*, weakly heterogeneous) and the *Single Knapsack Problem* (3D rectangular *SKP*, strongly heterogeneous).

Date: July 9, 2010.
Key words and phrases. Container loading, 3D Packing, genetic algorithm, multi-population, random keys.

Supported by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006.
AT&T Labs Research Technical Report.



Weakly heterogeneous CLP



Strongly heterogeneous CLP

Figure 1.1. Weakly and strongly heterogeneous CLPs

The CLP is NP-hard (Scheithauer, 1992). To date, heuristics have been the only viable alternative to find optimal or near-optimal packings. Many heuristic procedures have been proposed for solving the CLP. These include wall-building algorithms (George and Robinson, 1980, Loh and Nee, 1992), such as tabu search (Bortfeldt and Gehring, 1998, Bortfeldt et al., 2003), GRASP (Moura and Oliveira, 2005, Parreno et al., 2008b), simulated annealing (Mack et al., 2004), genetic algorithms (Gehring and Bortfeldt, 1997), tree search methods (Morabito and Arenales, 1994, Terno et al., 2000, Eley, 2002, Pisinger, 2002, Hifi, 2002, Fanslau and Bortfeldt, 2009), and hybrid heuristics (Bortfeldt and Gehring, 2001). In some case, the heuristics include parallelization (Bortfeldt and Gehring, 2002, Bortfeldt et al., 2003, Mack et al., 2004). All of these heuristics involve some type of neighborhood structure in which it moves from one feasible solution to another. The moves are usually applied on indirect representations of the solutions which change the order in which boxes are packed rather than their physical location.

Other authors have considered additional practical constraints. For instance, Davies and Bischoff (1999), Eley (2002), and Gehring and Bortfeldt (1997) take into account the weight distribution of cargo within a container. Bischoff (2006) examines the impact of varying the load-bearing strength. Several studies consider loading stability, including Bortfeldt and Gehring (2001), Bortfeldt et al. (2003), and Terno et al. (2000). Other container-related factors, such as orientation constraints (Gehring and Bortfeldt, 2002) and the grouping of boxes (Bischoff and Ratcliff, 1995, Takahara, 2005), have also been considered.

In this paper, we introduce a multi-population biased random-key genetic algorithm (BRKGA) for the CLP. The remainder of the paper is organized as follows. In Section 2, we formally define the problem In Section 3 we introduce the new approach, describing in detail the BRKGA, the placement strategy, and a novel procedure for joining maximal spaces. Finally, in Section 4, we report on computational experiments, and in Section 5 make concluding remarks.

2. The problem

The single container loading problem addressed in this paper can be applied to any mix of box types (i.e. from weakly to strongly heterogeneous sets of box types). Some practical constraints are taken into account. The problem may be stated as follows: A given 3D rectangular container C is to be loaded with a subset of a given set of rectangular boxes in such a way that all boxes are feasibly placed, the packed volume is maximized, and the constraints are met. A box is considered to be feasibly placed if it is arranged in such a way that it is parallel to the side walls of the container, does not overlap with another box, and lies completely inside the container. The dimensions of the rectangular container C are given as L (length), W (width), and H (height). The boxes to be loaded are categorized into K box types depending on their dimensions. For each box type k, there are N_k boxes with a length, width, and height of, respectively, l_k, w_k, and h_k, for $k = 1, 2, ..., K$.

Additional constraints, taken from the large number of constraints found in practice (cf. Bischoff and Ratcliff (1995)) are also considered. They are:

- *C1 - Orientation constraint:* Originally each box can be arranged in the container in a maximum of six *rotation variants*. However, for each box, up to five rotation variants may be prohibited by means of an orientation constraint. For example, some boxes require that one side be always on top.
- *C2 - Stability constraint:* To guarantee load stability, the bottom sides of all boxes not placed directly on the container floor must be completely supported by the top sides of one or more boxes.

3. Biased random-key genetic algorithm

We begin this section with an overview of the solution process. This is followed by a discussion of the biased random-key genetic algorithm, including detailed descriptions of the solution encoding and decoding, evolutionary process, fitness function, and parallel implementation.

3.1. **Overview.**

The new approach is based on a constructive heuristic algorithm which uses *layers* of boxes that may take the shape of a set columns or a set of rows. A layer is a rectangular arrangement of boxes of the same type, in rows and columns, filling one side of an empty space (see Figure 3.8). The management of the feasible placement positions is based on a list of empty *maximal-spaces* as described in Lai and Chan (1997). An 3D empty space in the container is maximal if it is not contained in any other other space in the container. Each time a layer is placed in an empty maximal-space, new empty maximal-spaces are generated. The new approach proposed in this paper combines a multi-population biased random-key genetic algorithm, a new placement strategy, and a novel procedure to join maximal-spaces having the same base level.

The role of the genetic algorithm is to evolve the encoded solutions, or *chromosomes*, which represent the *box type packing sequence* (*BTPS*) and the type of layer used to place each box type. For each chromosome, the following phases are applied to decode the chromosome:

1. *Decoding of the box type packing sequence.* This first phase decodes part of the chromosome into the *BTPS*. i.e. the sequence in which the box types are packed into the container.
2. *Decoding of layer types.* The second phase decodes part of the chromosome into the vector of layer types (*VLT*) used the by the placement procedure to select the type of layer used to pack boxes into the container.

(3) *Placement* procedure. The third phase makes use of the *BTPS* defined in phase 1 and the *VLT* obtained in phase 2 and constructs a packing of the boxes. In this phase, we develop a novel procedure, MaxJoin, which joins maximal-spaces having the same base level. This is done so the supporting area of the maximal-spaces is increased, increasing therefore the possibly of satisfying constraint *C2*.
(4) *Fitness evaluation.* The final phase computes the percentage volume packed, the fitness measure (quality measure) of the solution.

Figure 3.1 illustrates the sequence of steps applied to each chromosome generated by the BRKGA.

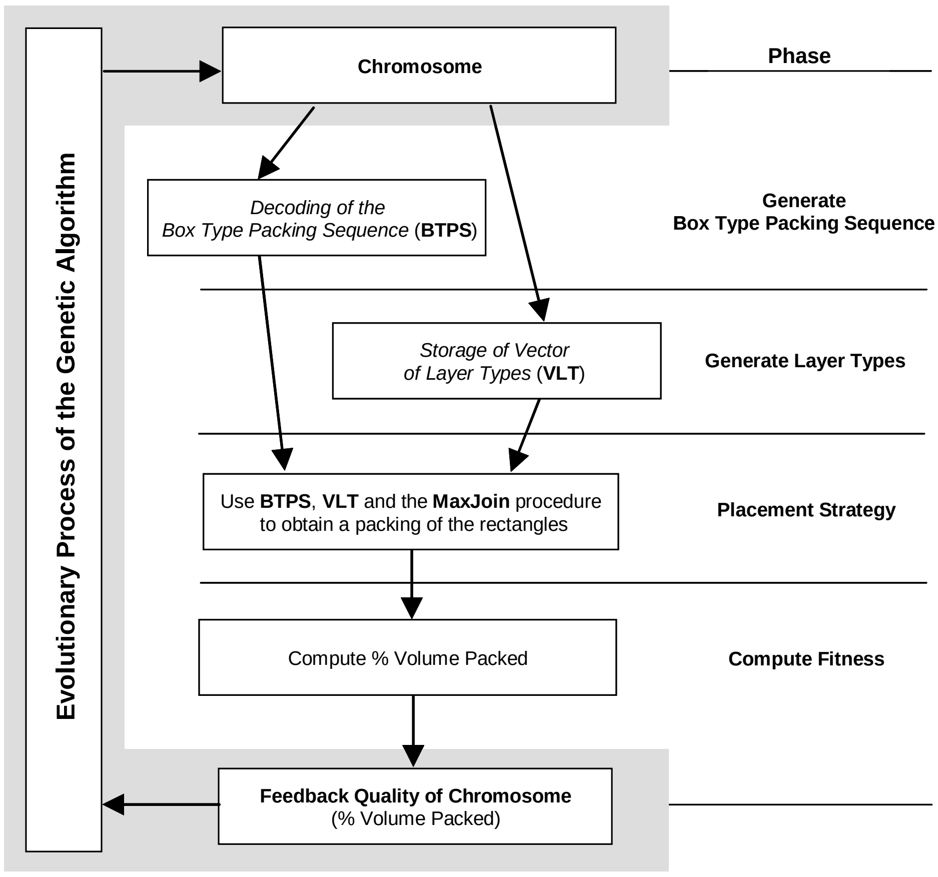

Evolutionary Process of the Genetic Algorithm
Chromosome
Phase
Decoding of the Box Type Packing Sequence (BTPS)
Generate Box Type Packing Sequence
Storage of Vector of Layer Types (VLT)
Generate Layer Types
Use BTPS, VLT and the MaxJoin procedure to obtain a packing of the rectangles
Placement Strategy
Compute % Volume Packed
Compute Fitness
Feedback Quality of Chromosome (% Volume Packed)

FIGURE 3.1. Architecture of the algorithm. Evolutionary process is on the left and decoder is on the right.

The remainder of this section describes in detail the genetic algorithm, the decoding procedure and the placement strategy.

3.2. Biased random-key genetic algorithm.

Genetic algorithms are adaptive methods that are used to solve search and optimization problems (Goldberg, 1989, Beasley et al., 1993). They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection, i.e. *survival of the fittest*, first clearly stated by Charles Darwin (1859). By mimicking this process, genetic algorithms, if suitably encoded, are able to *evolve* solutions to real-world problems. Before a genetic algorithm can be defined, an *encoding* (or *representation*) for the problem must be devised. A *fitness function*, which assigns a figure of merit to each encoded solution, is also required. During the run, parents are *selected* for reproduction and *recombined* to generate offspring. Figure 3.2 shows high-level pseudo-code for a standard genetic algorithm.

In genetic algorithms, a solution is encoded as a set of parameters, known as *genes*, joined together to form a string of values called a *chromosome*. In genetic terminology, the set of parameters represented by a particular chromosome is referred to as an *individual*. The *fitness* of an individual depends on its chromosome and is evaluated by the fitness function. During the reproductive phase, individuals are selected from the population and *recombined*, producing offspring, which comprise the next generation. Parents are randomly selected from the population using a scheme that favors fitter individuals. Once selected, the chromosomes of the two parents are *recombined*, typically using mechanisms of *crossover*. *Mutation* is usually applied to some individuals to guarantee population diversity.

```
procedure GENETIC-ALGORITHM
1  Generate initial population $P_0$;
2  Evaluate population $P_0$;
3  Initialize generation counter $g \leftarrow 0$;
4  while stopping criteria not satisfied repeat
5      Select some elements from $P_g$ to copy into $P_{g+1}$;
6      Crossover some elements of $P_g$ and put into $P_{g+1}$;
7      Mutate some elements of $P_g$ and put into $P_{g+1}$;
8      Evaluate new population $P_{g+1}$;
9      Increment generation counter: $g \leftarrow g+1$;
10 end while;
end GENETIC-ALGORITHM;
```

FIGURE 3.2. Pseudo-code of a standard genetic algorithm.

3.2.1. *Chromosome representation and decoding* . The heuristic described in this paper is a biased random-key genetic algorithm (Gonçalves and Resende, 2009). It uses a random-key alphabet comprised of random real numbers between 0 and 1. The evolutionary strategy used is similar to the one proposed by Bean (1994), the main difference occurring in the crossover operator. The important feature of this type of genetic algorithm is that all offspring formed by crossover are feasible solutions. This is accomplished by moving much of the feasibility issue into the objective function evaluation. If any random-key vector can be interpreted as a feasible solution, then any crossover vector is also feasible. Through the dynamics of the genetic algorithm, the system learns the relationship between random-key vectors and solutions with good objective function values.

A chromosome represents a solution to the problem and is encoded as a vector of random keys. In a direct representation, a chromosome represents a solution of the original problem, and is called *genotype*, while in an indirect representation it does not, and special procedures are needed to derive from it a solution called a *phenotype*. In the present context, the direct use of packing patterns as chromosomes is too complicated to represent and manipulate. In particular, it is difficult to develop corresponding crossover and mutation operations. Instead, solutions are represented indirectly by parameters that are later used by a decoding procedure to obtain a solution. To obtain the solution (phenotype) we use the placement strategy that we describe in Section 3.3.5.

Recall that there are K box types and that, for $k = 1, \dots, K$, at most N_k boxes of type k can be packed into the container. In the description of the genetic algorithm, we are given a total of $M = \sum_{k=1}^{K} N_k$ boxes. Each solution chromosome is made of $2M$ genes, i.e.

$$Chromosome = (\underbrace{gene_1, \dots, gene_M}_{\text{Box Type Packing Sequence}}, \underbrace{gene_{M+1}, \dots, gene_{2M}}_{\text{Vector of Layer Types}}).$$

The first M genes are used to obtain the *Box Type Packing Sequence* (*BTPS*), while the last M genes are used to obtain the *Vector of Layer Types* (*VLT*). The *BTPS* as well as the *VLT* are used by the placement strategy.

The decoding (mapping) of the first M genes of each chromosome into a *BTPS* is accomplished by sorting the genes and box types in ascending order. Figure 3.3 shows an example of the decoding process for the *BTPS*. In this example there are there four types of boxes with $N_1 = 2$, $N_2 = 3$, $N_3 = 1$, and $N_4 = 2$. According to the ordering obtained, the box types should be packed in the order 2, 4, 2, 1, 2, 1, 3, 4. The vector of layer types *VLT* is defined such that

$$VLT_i = Gene_{M+i},$$

i.e., each position $i = 1, \dots, M$ of VLT is populated with $Gene_{M+i}$.

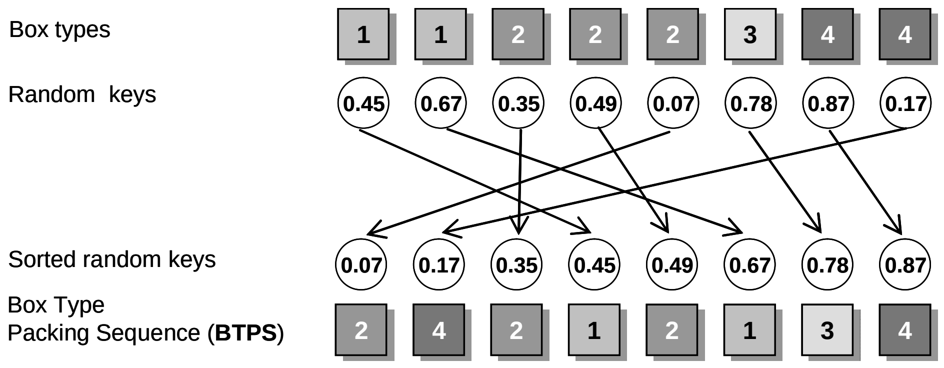


FIGURE 3.3. Chromosome decoding procedure for the Box Type Packing Sequence.

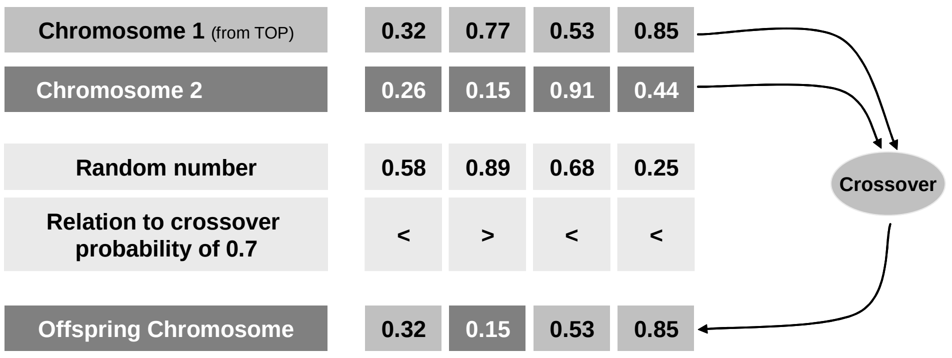
3.2.2. *Evolutionary process.* The population of random-key vectors is operated upon by a genetic algorithm to breed good solutions. Many variations of genetic algorithms, obtained by altering the reproduction, crossover, and mutation operators, have been proposed. The reproduction and crossover operators determine which parents will have offspring, and how the genetic material is exchanged between the parents to create those offspring. Mutation allows for random alteration of genetic material. Reproduction and crossover operators tend to increase the quality of the populations and force convergence. Mutation opposes convergence and replaces genetic material lost during reproduction and crossover.

In a random-key genetic algorithm, the population is initialized with random-key vectors whose components are random real numbers uniformly sampled from the interval $[0, 1]$. Reproduction is accomplished by first copying some of the best individuals from one generation to the next, in what is called an *elitist strategy* (Goldberg, 1989). The advantage of an elitist strategy over traditional probabilistic reproduction is that the best solution is monotonically improving from one generation to the next. The potential downside is population convergence to a local minimum. This can, however, be mitigated by an appropriate amount of mutation.

Parameterized uniform crossover (Spears and Dejong, 1991) is employed in place of the traditional one-point or two-point crossover. After two parents are chosen at random, one selected from the best (TOP in Figure 3.5) and the other from the full old population (including chromosomes copied to the next generation in the elitist pass), at each gene we toss a biased coin to select which parent will contribute the allele. Unlike Bean (1994), in a biased-random key genetic algorithm, we always select one parent from the set of elite solutions. Gonçalves and Resende (2009) show that, compared to the random-key GA of Bean, this change produces results with better quality and converges faster to good quality solutions. Figure 3.4 presents an example of the crossover operator. It assumes that a coin toss of heads selects the gene from the first parent, a tails chooses the gene from the second parent, and that the probability of tossing a heads is 0.7, i.e. the crossover probability $CProb = 0.7$. In Section 4 we describe how this value is determined empirically.

Rather than using the traditional gene-by-gene mutation with very small probability at each generation, a random-key GA adds a small set of new members to the population. These individuals, called *mutants*, are randomly generated from the same distribution as the initial population (see BOT in Figure 3.5). Like in standard mutation, the objective here is to prevent premature convergence of the population and leads to a simple statement of convergence. Figure 3.5 depicts the transitional process between two consecutive generations.

3.2.3. *Fitness function.* To feedback the quality of a solution to the evolutionary process, a measure of solution fitness, or quality measure, has to be defined. The natural fitness function for this type of problem is the *percent total packed volume* given by



Chromosome 1 (from TOP)	0.32	0.77	0.53	0.85
Chromosome 2	0.26	0.15	0.91	0.44
Random number	0.58	0.89	0.68	0.25
Relation to crossover probability of 0.7	<	>	<	<
Offspring Chromosome	0.32	0.15	0.53	0.85

FIGURE 3.4. Example of parameterized uniform crossover with crossover probability equal to 0.7. The offspring resembles parent 1 more than it does parent 2.

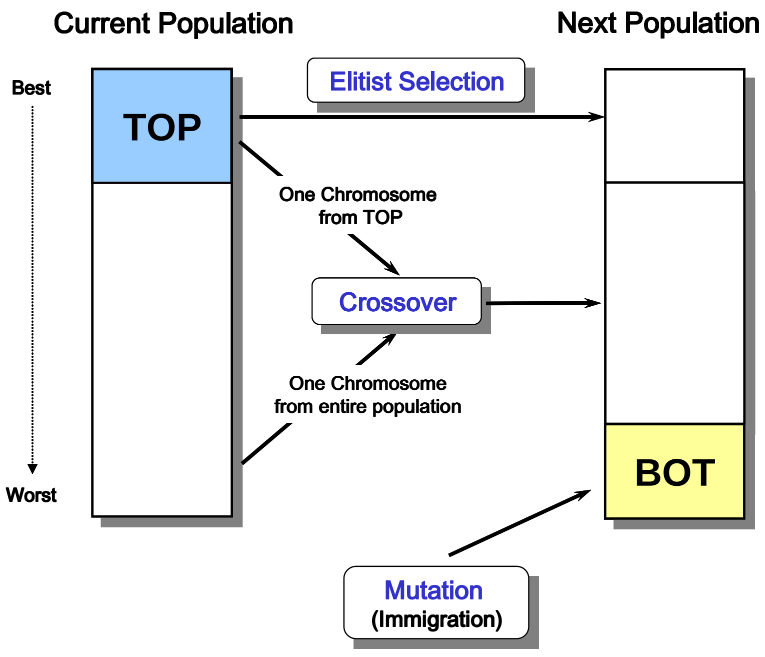


FIGURE 3.5. Transitional process between consecutive generations.

$$100\% \frac{\sum_{k=1}^{K} v_k \, NP_k}{L \times W \times H}$$

where NP_k is the number of boxes of type k packed in a solution, v_i is the volume of a box of type k and the denominator represents the volume of the container.

3.2.4. *Multi-population strategy.* In the multi-population strategy used in this paper, several populations are evolved independently in parallel. After a pre-determined number of generations, all the populations exchange good-quality chromosomes. When evaluating possible interchange strategies, we observed that exchanging too many chromosomes, or exchanging them too frequently, often lead to the disruption of the evolutionary process. With this in mind, we chose a strategy that, after a pre-determined number of generations, inserts the overall two best chromosomes (from the union of all populations) into all populations. In Section 4 we show how this choice was determined empirically.

3.3. **Placement strategy.**

3.3.1. *Maximal-spaces and the difference process.* While trying to place a box in the container we use a list S of empty maximal-spaces (*EMSs*), i.e. largest empty parallelepiped spaces available for filling with boxes. Maximal-spaces are represented by their vertices with minimum and maximum coordinates (x_i, y_i, z_i and X_i, Y_i, Z_i respectively). When searching for a place to pack a box we need to consider only the coordinates corresponding to the *EMS* vertices with minimum coordinates (x_i, y_i, z_i). To generate and keep track of the *EMSs*, we make use of the *difference process* (*DP*), developed by Lai and Chan (1997). Figure 3.6 depicts an example of the application of the *DP* process. In the example we assume that we have one box to be packed in the container (see Figure 3.6a. Since the container is empty, the box is packed at the origin of the container as shown in Figure 3.6b. In order to pack the next box, we first update the list S of empty maximal-spaces. Figure 3.6c shows the three new *EMSs* generated by the *DP* process. Every time a box is packed, we reapply the *DP* process to update list S before we pack the next box.

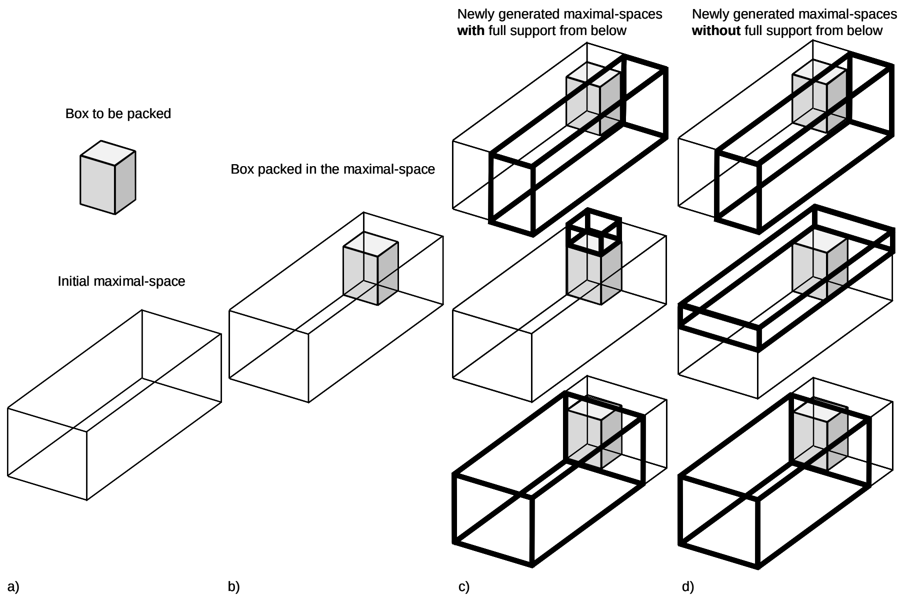

Newly generated maximal-spaces **with** full support from below
Newly generated maximal-spaces **without** full support from below
Box to be packed
Box packed in the maximal-space
Initial maximal-space
a)
b)
c)
d)

FIGURE 3.6. Example of difference process (*DP*) with and without full support from below.

There some real applications where full support from below (constraint *C2*) is not required. Figure 3.6d) presents the the newly generated maximal-spaces generated by the *DP* procedure when full support from below is enforced.

3.3.2. *The Back-Bottom-Left procedure.* Recall from Section 3.3.4 that x_i, y_i, z_i denote the minimum coordinates of EMS_i. The *Back-Bottom-Left* (*BBL*) procedure orders the *EMSs* in such a way that $EMS_i < EMS_j$ if $x_i < x_j$, or if $x_i = x_j$ and $z_i < z_j$, or if $x_i = x_j$, $z_i = z_j$, and $y_i < y_j$, and then chooses the first *EMS* in which the box type to be packed fits. Figure 3.7 shows pseudo-code for the *BBL* procedure.

3.3.3. *Layers of boxes.* The new loading approach is based on a constructive heuristic that uses layers of boxes. A *layer* is a rectangular arrangement of boxes of the same type, in rows and columns, filling one side of an empty maximal space.

To determine which layer type to use to pack a box type b_k we first fill the vector *Layers* with all the feasible layer-types that can be used to pack box type b_k into a predetermined empty maximal-space EMS^*. Each box type can have at most six rotation variants. For each variant,

```
procedure BBL( $b_k$, $S$ )
1  Let $b_k$ be a box of type $k$ to be packed in the container;
2  Let $N_S$ be the number of available $EMSs$ in $S$;
3  Initialize $X^* \leftarrow L,\ Y^* \leftarrow W,\ Z^* \leftarrow H$;
5  for $i = 1, \ldots, N_S$ do
6      Let $x(EMS_i)$ be the minimum $x$ coordinate of $EMS_i$;
7      Let $y(EMS_i)$ be the minimum $y$ coordinate of $EMS_i$;
7      Let $z(EMS_i)$ be the minimum $z$ coordinate of $EMS_i$;
8      if $b_k$ fits in $EMS_i$ then
9          if $x(EMS_i) \leq X$ or ($x(ERS_k) = X^*$ and $z(ERS_k) \leq Z^*$ ) or
·             ($x(ERS_k) = X^*$ and $z(ERS_k) = Z^*$ and $y(ERS_k) \leq Y^*$ ) then
10                $X^* \leftarrow x(ERS_k)$, $Z^* \leftarrow z(ERS_k)$, $Y^* \leftarrow y(ERS_k)$ ;
11                $EMS^* = EMS_i$;
12         end if
13     end if
14 end for
15 Return $EMS^*$;
end BBL;
```

FIGURE 3.7. Pseudo-code of the Back-Bottom-Left (BBL) procedure.

we can have at most six types of layers. Therefore, we have at most 36 layers types. Figure 3.8 shows all possible six layers types that can be defined for one of the six box type variants and a empty-maximal-space where the layers can be packed.

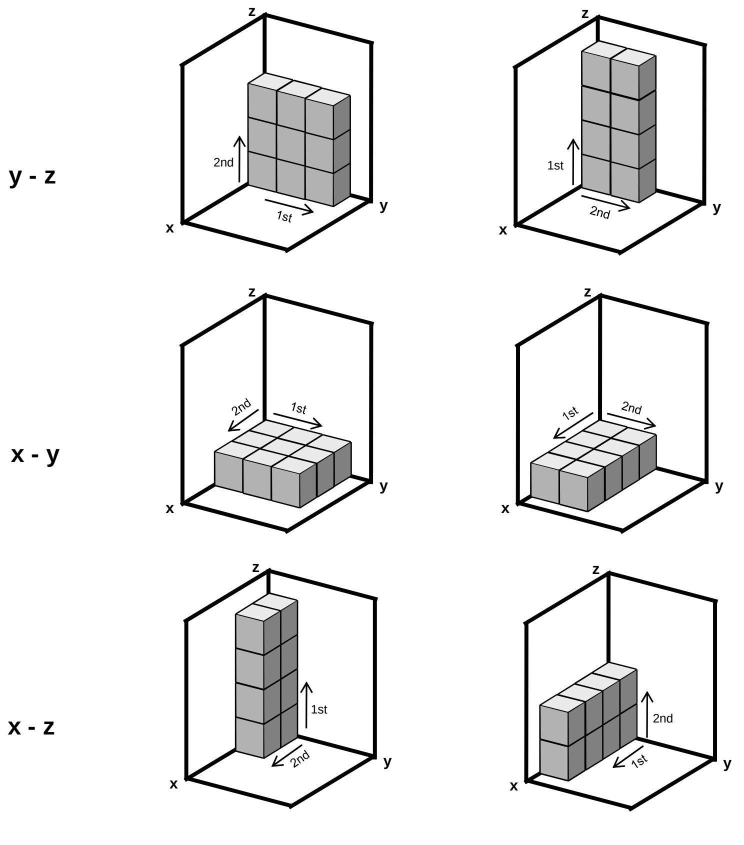


FIGURE 3.8. Example of the 6 different feasible layers types for a box type rotation variant.

3.3.4. *Joining maximal spaces.* The *DP* procedure presented in Section 3.3.1 generates new *EMSs* each time a box is added to the container. However, when a new box is added, the supporting area of some of the previously generated *EMSs* can sometimes increase. Since the *DP* process does not take this into account, in such situations the possibility of satisfying constraint *C2* (full supporting from below) is reduced. In this section, we develop a novel procedure we call MaxJoin which joins maximal-spaces having the same supporting area height.

To illustrate MaxJoin, we use the example depicted in Figure 3.9, where we assume that the box labeled x was the last one to be packed (see Figure 3.9a). Figure 3.9b shows all packed boxes that have the same height as box x. Figure 3.9c shows a top-down view of the supporting area defined by the boxes. In the remainder of this section, we restrict ourselves only to the top-down view since the heights of the *EMSs* are equal and known.

The MaxJoin procedure consists of two main steps in which the *DP* procedure is applied twice to obtain the desired *EMSs*. In the first step, the *DP* procedure in applied to subtract from the container the spaces corresponding to the boxes (see Figure 3.10a). Note that the resulting *EMSs* denoted by 1, 2, 3, and 4 in Figure 3.10a correspond to the complement of the *EMSs* that we seek. In the second step, we apply the *DP* procedure to subtract from the container the final *EMSs* obtained in the first step. The resulting sought *EMSs* are shaded in Figure 3.10b.

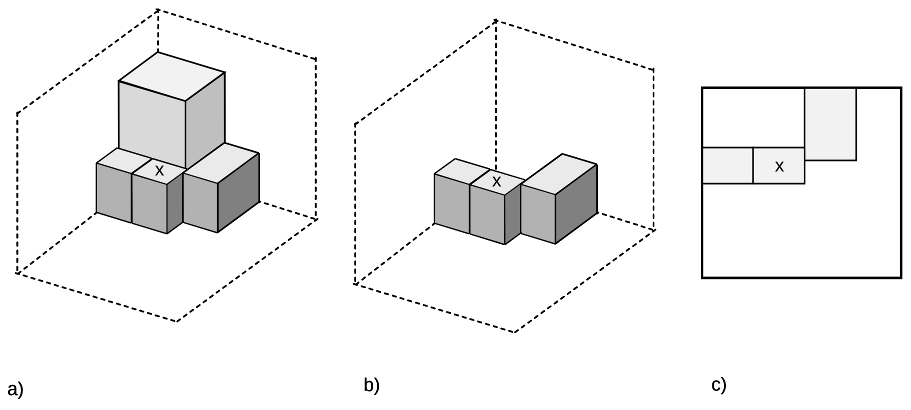


FIGURE 3.9. Example where the empty maximal-spaces are not joined by *DP*.

3.3.5. *Placement procedure.* The placement procedure follows a sequential process which tries to pack a box or a layer of boxes at each stage. The procedure combines four elements: the list $BTPS$ of box types defined by the genetic algorithm, a list S of empty maximal spaces, initially containing only container C, the *BBL* procedure, and the vector of layer types (VLT) also defined by the GA. Each stage is comprised of the following five main steps:

(1) Box type selection;
(2) Maximal space selection;
(3) Layers type selection;
(4) Layer packing;
(5) State information update.

The pseudo-code of the placement procedure is given in Figure 3.3.5. The box type selection step consists in choosing from $BTPS$ the first box type k^* which has not yet been used (lines 9 to 11 of the pseudo-code). The maximal space selection is carried out by the *BBL* procedure and the list S to produce EMS^* (lines 12 to 13 of the pseudo-code). If a maximal space was found in the previous step, then the layer selection uses *VLT,* the vector $Layers$, and all the possible layer types of box type k^* that can be packed into EMS^*, to obtain the selected layer type $Layers^*$ (lines 17 to 19 of the pseudo-code). The layer packing step consists in packing $Layer*$ into $EMS*$ (lines 17-19 of the pseudo-code). The final step, state information update, consists in updating the remaining quantities of the box type packed k^* and updating list S, using the *DP* and MaxJoin procedures, as well as some flags (*Skip* and *Placed*) (lines 21 to 31 of the pseudo-code).

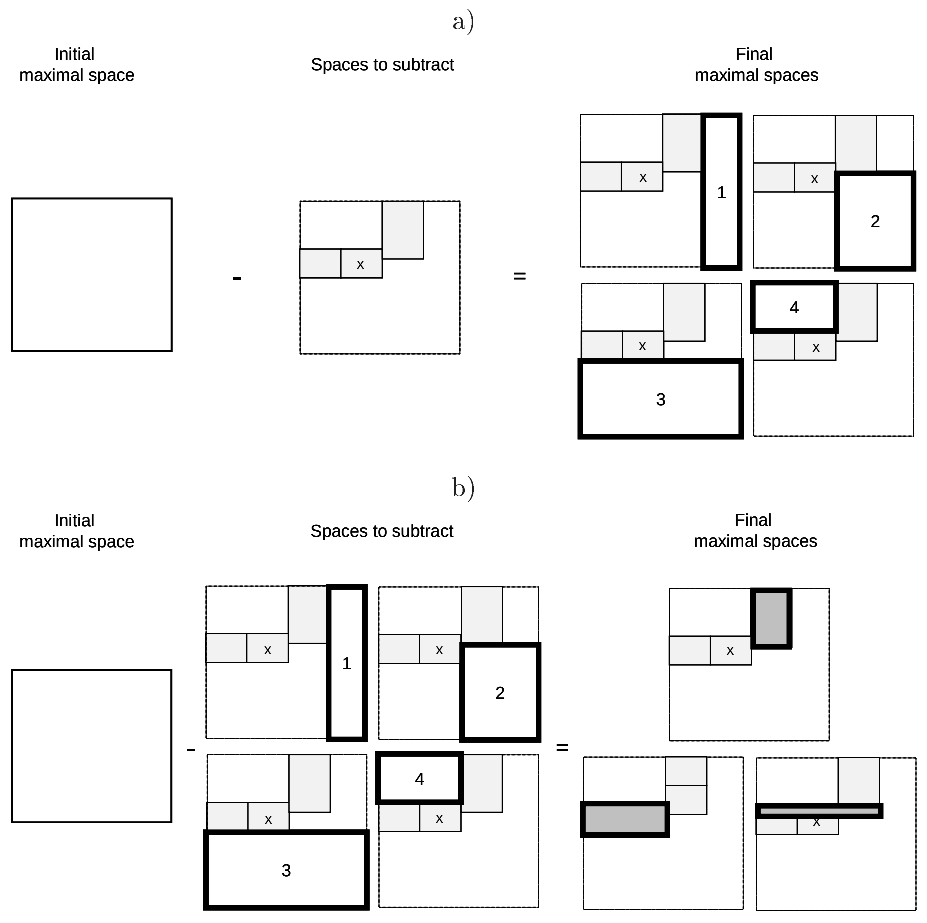


FIGURE 3.10. Joining spaces with the MaxJoin procedure.

3.4. **Parallel implementation.** We limit parallelization only to the task that performs the evaluation of the chromosome fitness since it is the most time consuming. The tasks related with the GA logic were not parallelized since they consume very little time. This type of parallelization is easy to implement and in multi-core CPUs allows for a large reduction in computational times (almost a linear speed-up with the number of cores). The parallel implementation of our heuristic was done using the OpenMP Application Program Interface (API) which supports multi-platform shared-memory parallel programming in C/C++.

4. NUMERICAL EXPERIMENTS

In this section we report on results obtained on a set of experiments conducted to evaluate the performance of the multi-population biased random key genetic algorithm for a container loading problem (BRKGA-CLP) proposed in this paper.

4.1. **Benchmark algorithms.** We compare BRKGA-CLP with the 13 approaches listed in Table 1. These approaches comprise the most effective to date.

4.2. **Test problem instances.** The effectiveness of BRKGA-CLP is evaluated by solving the complete set of 1500 problems suggested by Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) which range from weakly heterogeneous to strongly heterogeneous cargo.

The instances are divided in 15 test cases, each with 100 instances, and are referred to as BRD_01 to BRD_15. The number of different box types in each case are 3, 5, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, and 100. The structure of each problem changes gradually from weakly heterogeneous to strongly heterogeneous according to the decreasing average number of boxes per type.

```
procedure PLACEMENT ($BTPS$, $VLT$, $FullSupport$)
1  Let $Placed_i$ be a flag that indicates whether the box type given by $BTPS(i)$
·          has already been used to pack a box type or not;
2  Let $S$ be the list of available empty $EMSs$;
3  Let $QtRemain_k$ be the remaining quantity of unpacked boxes of type $k$;
4  Let $Skip_k$ be a flag that indicates whether the box type $k$ should
·          be skipped or not when searching for the next box type to pack;

   // ** Initialization
5  $S \leftarrow Empty\ container$;
6  $QtRemain_k \leftarrow N_k$, $Skip_k \leftarrow False$, for all $k$;
7  $Placed_i \leftarrow False$ for all $i$;

8  do while (There exits at least one $k$ for which $Skip_k = False$);
      // ** Box type selection
9     $i^* \leftarrow 0$;
10    Let $i^*$ be the first index $i$ in $BTPS$ for which $Placed_i = False$ and $Skip_{BTPS(i)} = False$;
11    Let $k*$ be box type corresponding to $BTPS(i^*)$;

      // ** Maximal space selection
12    $EMS^* \leftarrow 0$;
13    Let $EMS^*$ be the $EMS$ in $S$ in which a box of type $k^*$
·            is placed when the Back-Bottom-Left placement heuristic is applied;

14    if $EMS^* = 0$ then //  no $EMS$ was found ;
15       $Skip_{k^*} = False$; // box type $k^*$ is not packable
16    else if ;
         // ** Layer type selection
17       According to $QtRemain_{k^*}$ fill the vector $Layers$
·               with all the layer-types packable into maximal-space $EMS^*$;
18       Let $MaxLayers$ be number of layers in vector $Layers$;
19       Let $Layer^* = Layers\,(\lceil VLT(i^*) \times MaxLayers\rceil)$ be the layer type selected
·               for placing the box type $k^*$, ($\lceil x\rceil$denotes the minimum integer greater than $x$);

         // ** Layer packing
20       Pack $Layer^*$ at the origin of maximal space $EMS*$;

         // ** Information update
21       Let $nBox$ be the number of boxes of type $k^*$contained in $Layer^*$;
22       $QtRemain_{k^*} = QtRemain_{k^*} - nBox$;
23       if $QtRemain_{k^*} = 0$ then $Skip_{k*} = True$; // no more boxes of type $k^*$to pack
24       $Placed_{i^*} = True$;
25       Update $S$ using the DP procedure of Lai and Chan (1997);
26       if $FullSupport$ then
27          Update $S$ by applying the MaxJoin procedure to all the $EMSs$ with
·                  origin height equal to the top height of the $Layer^*$after being packed;
28          $Skip_k \leftarrow False$ for all $\{k\,|\,QtRemain_k > 0\}$;
29       end if
30    end if
31 end do
end PLACEMENT;
```

FIGURE 3.11. Pseudo-code for the PLACEMENT procedure.

Table 1. Efficient approaches used for comparison.

Approach	Source of approach	Type of method
T_BB	Terno et al. (2000)	Branch and Bound
BG_GA	Bortfeldt and Gehring (2001)	GA
BG_PGA	Bortfeldt and Gehring (2002)	Parallel GA
E_TRS	Eley (2002)	Tree Search (TRS)
L_GH	Lim et al. (2003)	Greedy Heuristic
B_PTS	Bortfeldt et al. (2003)	Parallel Tabu Search (TS)
B_NMP	Bischoff (2006)	Nelder-Mead Proc.
M_SATS	Mack et al. (2004)	Parallel SA/TS
MO_GR	Moura and Oliveira (2005)	GRASP
P_GR	Parreno et al. (2008b)	GRASP
P_VNS	Parreno et al. (2008a)	VNS
FB_TRS	Fanslau and Bortfeldt (2009)	TRS
HH_HBS	He and Huang (2010)	Heuristic Beam Search

In test case BRD_01 there are on average 50.15 boxes for each box type, whereas in test case BRD_15 the average number is only 1.33. Each individual instance never exceeds the volume of the container and the average of available cargo is over 99.46% of the capacity of the container. The dimensions of the boxes were generated independently of the dimensions of the container, therefore there is no guarantee that all the boxes will fit into the container. The percentage given should be seen as a loose upper bound on the volume of the container attainable by an optimal packing.

Each instance includes the orientation constraint (*C1*), which prohibits the use of certain larger side dimension as height dimension.

4.3. **GA configuration.** Configuring genetic algorithms is oftentimes more an art form than a science. In our past experience with genetic algorithms based on the same evolutionary strategy (see Gonçalves and Almeida (2002), Ericsson et al. (2002), Gonçalves and Resende (2004), Gonçalves et al. (2005), Buriol et al. (2005), Buriol et al. (2007), Gonçalves (2007), Gonçalves et al. (2009), Gonçalves et al. (2009), Gonçalves and Resende (2009) and Gonçalves and Resende (2009)), we obtained good results with values of TOP, BOT, and Crossover Probability (CProb) in the intervals shown in Table 2.

Table 2. Range of Parameters in past implementations.

Parameter	Interval
TOP	0.10 - 0.25
BOT	0.15 - 030
Crossover Probability (CProb)	0.70 - 0.80

For the population size, we have obtained good results by indexing it to the dimension of the problem, i.e. we use small size populations for small problems and larger populations for larger problems. With this in mind, we conducted a small pilot study to obtain a reasonable configuration. We tested all the combinations of the following values:

- $TOP \in \{0.10, 0.15, 0.20, 0.25\}$;
- $BOT \in \{0.15, 0.20, 0.25, 0.30\}$;
- $CProb \in \{0.70, 0.75, 0.80\}$;
- *Population size* with 10, 15, 20, and 25 times the number of rectangles in the problem instance.

For each of the 192 possible configurations, we made three independent runs of the algorithm (with three distinct random number generator seeds) and computed the average total value. The configuration that minimized the sum, over the pilot problem instances, was $TOP = 15\%$, $BOT =$

15%, *CProb* = 0.7, and *Population size* = 20 times the number of rectangles in the problem instance.

After some experimentation with the problem instances in a pilot study we came to the conclusion that using three parallel populations and exchanging information every 15 generations was a reasonable configuration for this type of problem.

The configuration presented in Table 3 was held constant for all experiments and all problems instances. The computational results presented in the next section demonstrate that this configuration not only provides excellent results in terms of solution quality but is also very robust.

Table 3. Configuration used on all runs in the computational experiments.

Population size:	20 × *number of input boxes*
Crossover probability:	0.7
TOP:	The 15 % most fit chromosomes from the previous generation are copied to the next generation
BOT:	The 15 % least fit chromosomes from the previous generation are replaced with randomly generated chromosomes
Number of population:	3
Exchange information btw pops:	Every 15 generations
Fitness:	Maximize % total packed volume
Stopping Criterion:	Stop after 500 generations

4.4. **Computational results.** Algorithm *BRKGA-CLP* was implemented in C++. The computational experiments were carried out on a computer with a AMD 2.2 GHz Opteron 6-core CPU with the Linux (Fedora release 12) operating system.

All computational results show average values for the 100 instances of each test case. All tests where performed using the configuration summarized in Table 3.

For comparison purposes and because some authors report computational results where the support constraint (*C2*) is not enforced we present results for two versions of our approach: version *BRKGA-CLP-S* (supported) that enforces the support constraint (*C2*) and version *BRKGA-CLP-U* (unsupported) which does not.

We note that some of approaches in Table 1 report results only for the first seven sets of weakly heterogeneous instances, BRD_01–BRD_07.The complete computational results appear in Tables 4 and 5 for versions *BRKGA-CLP-S* and *BRKGA-CLP-U*, respectively.

As can be observed from Tables 4 and 5 both versions of *BRKGA-CLP* obtain the best overall results (for *BRKGA-CLP-S*: Mean 01-07 = 94.53%, Mean 08-15 = 90.23%, Mean 01-15 = 92.24% while for *BRKGA-CLP-U*: Mean 01-07 = 95.74%, Mean 08-15 = 93.49%, Mean 01-15 = 94.54%). The second best approach is FB_TRS and it obtains overall means 01-15 which are 0.26% and 0.59% worst than *BRKGA-CLP*. *BRKGA-CLP* is only outperformed by *FB_TRS* for test case BRD_01 when full support is enforced. For all the other test cases *BRKGA-CLP* finds better average solutions than any of the other approaches.

In terms of computational times we cannot make any fair and meaningful comments since all the other approaches were implemented and tested on computers with different computing power. Instead, we limit ourselves to reporting the average running times for the best three approaches.

5. Concluding remarks

In this paper we addressed the Single Container Loading Problem (CLP) where several rectangular boxes of different sizes are to be loaded into a single rectangular container. The approach uses a maximal-space representation to manage the free spaces in the container. The approach hybridizes a novel placement procedure with a multi-population genetic algorithm based on random keys. The genetic algorithm is used to evolve the order in which the box types are loaded into the container and to determine the corresponding types of layers used in the packing. A heuristic procedure is used to determine the maximal space where each box is placed and a novel procedure is developed for joining free spaces for the case where full support from below is required.

TABLE 4. Performance comparison of *BRKGA-CLP* with other approaches when support constraint (*C2*) is enforced.

Test case	T_BB	BG_GA	BG_PGA	E_TRS	B_NMP	MO_GR	FB_TRS (packing variant)	BRKGA CLP-S
BRD_01	89.9	87.81	88.10	88.0	89.39	89.07	**94.51**	94.34
BRD_02	89.6	89.40	89.56	88.5	90.26	90.43	94.73	**94.88**
BRD_03	89.2	90.48	90.77	89.5	91.08	90.86	94.74	**95.05**
BRD_04	88.9	90.63	91.03	89.3	90.90	90.42	94.41	**94.75**
BRD_05	88.3	90.73	91.23	89.0	91.05	89.57	94.13	**94.58**
BRD_06	87.4	90.72	91.28	89.2	90.70	89.71	93.85	**94.39**
BRD_07	86.3	90.65	91.04	88.0	90.44	88.05	93.20	**93.74**
BRD_08	-	89.73	90.26	-	-	86.13	92.26	**92.65**
BRD_09	-	89.06	89.50	-	-	85.08	91.48	**91.90**
BRD_10	-	88.40	88.73	-	-	84.21	90.86	**91.28**
BRD_11	-	87.53	87.87	-	-	83.98	90.11	**90.39**
BRD_12	-	86.94	87.18	-	-	83.64	89.51	**89.81**
BRD_13	-	86.25	86.70	-	-	83.54	88.98	**89.27**
BRD_14	-	85.55	85.81	-	-	83.25	88.26	**88.57**
BRD_15	-	85.23	85.48	-	-	83.21	87.57	**87.96**
Avg. 01-07	88.51	90.06	90.43	88.79	90.55	89.73	94.22	**94.53**
Avg. 08-15	-	87.34	87.69	-	-	84.13	89.88	**90.23**
Avg. 01-15	-	88.61	88.97	-	-	86.74	91.91	**92.24**

Note: The best values appear in **bold.**

TABLE 5. Performance comparison of *BRKGA-CLP* with other approaches when support constraint (*C2*) is not enforced.

Test case	L_GH	B_PTS	M_SATS	P_GR (200000)	P_GR (5000)	P_VNS	FB_TRS (cutting variant)	HH_HBS	BRKGA CLP-U
BRD_01	88.70	93.52	93.70	93.85	93.27	94.93	95.05	87.54	**95.28**
BRD_02	88.17	93.77	94.30	94.22	93.38	95.19	95.43	89.12	**95.90**
BRD_03	87.52	93.58	94.54	94.25	93.39	94.99	95.47	90.32	**96.13**
BRD_04	87.58	93.05	94.27	94.09	93.16	94.71	95.18	90.57	**96.01**
BRD_05	87.30	92.34	93.83	93.87	92.89	94.33	95.00	90.78	**95.84**
BRD_06	86.86	91.72	93.34	93.52	92.62	94.04	94.79	90.91	**95.72**
BRD_07	87.15	90.55	92.50	92.94	91.86	93.53	94.24	90.88	**95.29**
BRD_08	-	-	-	-	91.02	92.78	93.70	90.85	**94.76**
BRD_09	-	-	-	-	90.46	92.19	93.44	90.64	**94.34**
BRD_10	-	-	-	-	89.87	91.92	93.09	90.43	**93.86**
BRD_11	-	-	-	-	89.36	91.46	92.81	90.23	**93.60**
BRD_12	-	-	-	-	89.03	91.20	92.73	89.97	**93.22**
BRD_13	-	-	-	-	88.56	91.11	92.46	89.88	**92.99**
BRD_14	-	-	-	-	88.46	90.64	92.40	89.67	**92.68**
BRD_15	-	-	-	-	88.36	90.38	92.40	89.54	**92.46**
Avg. 01-07	87.61	92.70	93.78	93.82	92.94	94.53	95.02	90.02	**95.74**
Avg. 08-15	-	-	-	-	89.39	91.46	92.88	90.15	**93.49**
Avg. 01-15	-	-	-	-	91.05	92.89	93.88	90.09	**94.54**

Note: The best values appear in **bold.**

Two versions of the approach (with and without enforcement of full support from below) are is extensively tested on the complete set of problems of Bischoff and Ratcliff (1995) and Davies and

Table 6. Computational times (s) for best three approaches.

	Avg. Time (s) for test cases BRD_01 - BRD_15		
	Parreno et al. (2008a)	Fanslau and Bortfeldt (2009)	BRKGA-CLP
Supported	-	320	232
Unsupported	238	320	147

Bischoff (1999) which range from weakly to strongly heterogeneous cargo and compared with 13 other solution techniques. The experimental results validate the excellent quality of the solutions and the effectiveness of the proposed algorithm.

Acknowledgments

This work has been supported by funds granted by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/72244/2006.

References

Bean, J. C. (1994). Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160.

Beasley, D., Bull, D. R., and Martin, R. R. (1993). An overview of genetic algorithms: Part 1, Fundamentals. *University Computing*, 15:58–69.

Bischoff, E. (2006). Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168(3):952–966.

Bischoff, E. E. and Ratcliff, M. S. W. (1995). Issues in the Development of Approaches to Container Loading. *Omega, International Journal of Management Science*, 23(3):377–390.

Bortfeldt, A. and Gehring, H. (1998). A Tabu Search Algorithm for Weakly Heterogeneous Container Loading Problems. *OR Spektrum*, 20(4):237–250 (in German).

Bortfeldt, A. and Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161.

Bortfeldt, A. and Gehring, H. (2002). A Parallel Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, 9(4):497–511.

Bortfeldt, A., Gehring, H., and Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29(5):641–662.

Buriol, L. S., Resende, M. G. C., Ribeiro, C. C., and Thorup, M. (2005). A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46:36–56.

Buriol, L. S., Resende, M. G. C., and Thorup, M. (2007). Survivable IP network design with OSPF routing. *Networks*, 49:51–64.

Darwin, C. R. (1859). *On the origin of species through natural selection.* John Murray, London.

Davies, A. P. and Bischoff, E. E. (1999). Weight distribution considerations in container loading. *European Journal of Operational Research*, 114(3):509–527.

Eley, M. (2002). Solving Container Loading Problems by Block Arrangement. *European Journal of Operational Research*, 141(2):393–409.

Ericsson, M., Resende, M. G. C., and Pardalos, P. M. (2002). A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333.

Fanslau, T. and Bortfeldt, A. (2009). A Tree Search Algorithm for Solving the Container Loading Problem. *INFORMS Journal on Computing.*

Gehring, H. and Bortfeldt, A. (1997). A Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, pages 401–418.

Gehring, H. and Bortfeldt, A. (2002). A parallel genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 9(4):497–511.

George, A. J. and Robinson, D. F. (1980). A Heuristic for Packing Boxes into a Container. *Computers and Operations Research*, 7(3):147–156.

Goldberg, D. E. (1989). *Genetic algorithms in search optimization and machine learning.* Addison-Wesley.

Gonçalves, J. F. (2007). A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183:1212–1229.

Gonçalves, J. F. and Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8:629–642.

Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95.

Gonçalves, J. F. and Resende, M. G. C. (2004). An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering*, 47:247–273.

Gonçalves, J. F. and Resende, M. G. C. (2009). Biased random key genetic algorithms for combinatorial optimization. Technical report, AT&T Labs Research Technical Report, Florham Park, NJ 07733 USA.

Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2009). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189:1171–1190.

Gonçalves, J. F. and Resende, M. G. C. (2009). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, In press.

He, K. and Huang, W. (2010). Solving the single-container loading problem by a fast heuristic method. *Optimization Methods and Software*, 25(2):263–277.

Hifi, M. (2002). Approximate algorithms for the container loading problem. *International Transactions in Operations Research*, 9:747–774.

Lai, K. K. and Chan, J. W. M. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, 32:115–127.

Lim, A., Rodrigues, B., and Wang, Y. (2003). A multi-faced buildup algorithm for three-dimensional packing problems. *Omega*, 31(6):471–481.

Loh, T. H. and Nee, A. Y. C. (1992). A packing algorithm for hexahedral boxes. In *Proceedings of the Conference of Industrial Automation*, pages 115–126.

Mack, D., Bortfeldt, A., and Gehring, H. (2004). A Parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research*, 11:511–533.

Morabito, R. and Arenales, M. (1994). An AND/OR-graph approach to the container loading problem. *International Transactions in Operational Research*, 1(1):59–73.

Moura, A. and Oliveira, J. F. (2005). A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57.

Parreno, F., Alvarez-Valdes, R., Oliveira, J. F., and Tamarit, J. M. (2008a). Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics*.

Parreno, F., Alvarez-Valdes, R., Tamarit, J. M., and Oliveira, J. F. (2008b). A Maximal-Space Algorithm for the Container Loading Problem. *INFORMS JOURNAL ON COMPUTING*, 20(3):412–422.

Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal of Operational Research*, 141:143–153.

Scheithauer, G. (1992). Algorithm for the container loading problem. In *Operational Research Proceedings*, pages 445–452.

Spears, W. M. and Dejong, K. A. (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236.

Takahara, S. (2005). Loading problem in multiple containers and pallets using strategic search method. In *Modeling Decisions for Artificial Intelligence, Proceedings Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, Heidelberg,*, volume 3558, pages 448–456.

Terno, J., Scheithauer, G., Sommerweiss, U., and Riehme, J. (2000). An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research*, 123(2):372–381.

Wäscher, G., Haussner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130.

LIAAD, Faculdade de Economia do Porto, Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal

E-mail address: jfgoncal@fep.up.pt

Algorithms and Optimization Research Department, AT&T Labs Research,, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA

E-mail address: mgcr@research.att.com