Computers & Operations Research 39 (2012) 179–190

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A parallel multi-population biased random-key genetic algorithm for a container loading problem

José Fernando Gonçalves [a], Mauricio G.C. Resende [b,*]

[a] LIAAD, Faculdade de Economia do Porto, Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200 464 Porto, Portugal
[b] Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932, USA

A R T I C L E I N F O

Available online 1 April 2011

Keywords:
Container loading
3D packing
Genetic algorithm
Multi-population
Random keys

A B S T R A C T

This paper presents a multi-population biased random-key genetic algorithm (BRKGA) for the single container loading problem (3D-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 [1] and Davies and Bischoff [2] and is compared with 13 other approaches. The test set consists of 1500 instances from weakly to strongly heterogeneous cargo. The computational experiments demonstrate that not only the approach performs very well in all types of instance classes but also it obtains the best overall results when compared with other approaches published in the literature.

1. Introduction

The *single container loading problem* (3D-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. Fig. 1 shows that 3D-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. 3D-CLPs with relatively few box types are often referred to as *weakly heterogeneous* [1]. According to the typology of Wäscher et al. [3] for cutting and packing problems, the heuristic for the 3D-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).

In this paper we present a novel multi-population biased random-key genetic algorithm (BRKGA) for the 3D-CLP. 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. Two versions of the approach (with and without enforcement of full support from below) are extensively tested on the complete set of problems of Bischoff and Ratcliff [1] and Davies and Bischoff [2] containing 1500 instances, ranging from weakly to strongly heterogeneous cargo and is compared with 13 other solution techniques.

The computational experiments demonstrate that not only the approach performs very well in all types of instance classes considered but also it obtains the best overall results when compared with other approaches published in the literature. It has improved the overall averages from 93.88% and 91.91% to 94.54% and 92.24% for the unsupported and supported cases, respectively.

The remainder of the paper is organized as follows. In Section 2, we formally define the problem. In Section 3, we present a literature review. In Section 4, we introduce the new approach, describing in detail the BRKGA, the novel placement strategy, and the novel procedure used for joining maximal-spaces. Finally, in Section 5, we report on computational experiments, and in Section 6 make concluding remarks.

* Corresponding author.
E-mail addresses: jfgoncal@fep.up.pt (J.F. Gonçalves), mgcr@research.att.com (M.G.C. Resende).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.03.009

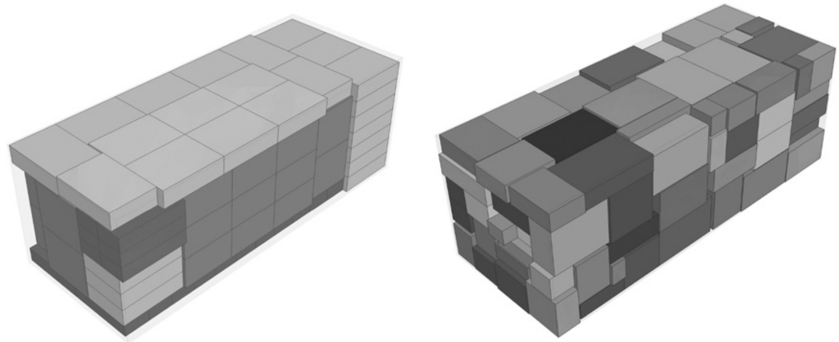

Fig. 1. Weakly and strongly heterogeneous 3D-CLPs.

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 $k=1,2,\ldots,K$.

Additional constraints, taken from the large number of constraints found in practice (cf. [1]) 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. Literature review

The 3D-CLP is NP-hard [4]. To date, only a few exact methods have been suggested in the literature. Fekete and Schepers [5] present a general framework for the exact solution of multi-dimensional packing problems. Martello et al. [6] develop an exact branch-and-bound method (B&B) for the 3D-CLP.

Heuristics have been the only viable alternative to find optimal or near-optimal packings. Many heuristic procedures have been proposed for solving the 3D-CLP. Fanslau and Bortfeldt [7] classify approaches for the 3D-CLP according to packing heuristic and method type. They group packing heuristics as wall-building, stack-building, horizontal layer-building, block-building, and guillotine cutting.

(1) *Wall-building approaches* fill the container with vertical layers ("Walls"). Among others it has been used by George and Robinson [8], Loh and Nee [9], Bortfeldt and Gehring [10] and Pisinger [11].
(2) *Stack-building approaches* fill the container with stacks, which are arranged on the floor of the container in a way that saves the most space. The heuristic of Bischoff and Ratcliff [1] and the genetic algorithm of Gehring and Bortfeldt [12] are examples of the use of this approach.
(3) *Horizontal layer-building approaches* fill the container from bottom to top using horizontal layers that are intended to cover the largest possible part of the load surface underneath. This approach has been implemented in Bischoff et al. [13] and Terno et al. [14].
(4) *Block-building approaches* fill the container with cuboid blocks of boxes. The tree-search method of Eley [15], the tabu search method of Bortfeldt et al. [16], and the hybrid simulated annealing and tabu search method of Mack et al. [17] are examples of the use of this approach.
(5) *Guillotine-cutting approaches are* based on a slicing tree representation of a packing plan. Each slicing tree corresponds to a successive segmentation of the container into smaller pieces by means of guillotine cuts, whereby the leaves correspond to the boxes to be packed. The graph-search method of Morabito and Arenales [18] is based on this approach.

Fanslau and Bortfeldt [7] categorize solution methods as meta-heuristics, tree-search methods, and conventional heuristics.

(1) *Metaheuristics* search strategies have been the preferred method in the last 10 years. These include the tabu search approaches (TS) of Bortfeldt et al. [16], the simulated annealing methods (SA) of Mack et al. [17], the genetic algorithms (GA) of Gehring and Bortfeldt [12,19] and Bortfeldt and Gehring [10], the approach of Bischoff [20], based on the Nelder and Mead algorithm, and the greedy randomized adaptive search procedures (GRASP) of Moura and Oliveira [21] and Parreno et al. [22].
(2) *Tree-search methods* or graph-search methods have been successfully applied to the 3D-CLP by Morabito and Arenales [18], Eley [15], Hifi [23], Pisinger [11], and Fanslau and Bortfeldt [7].
(3) *Conventional heuristics* incorporate construction methods and improvement methods. Examples include, e.g. papers by Bischoff et al. [13], Bischoff and Ratcliff [1], and Lim et al. [24].

Other authors have considered additional practical constraints. For instance, Davies and Bischoff [2], Eley [15], and Gehring and Bortfeldt [12] take into account the weight distribution of cargo within a container. Bischoff [25] examines the impact of varying the load-bearing strength. Several studies consider loading stability, including Bortfeldt and Gehring [10], Bortfeldt et al. [16], and Terno et al. [14]. Other container-related factors, such as orientation constraints [19] and the grouping of boxes [1,26], have also been considered.

4. Biased random-key genetic algorithm

We begin this section with an overview of the proposed 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.

4.1. Overview

The new approach is based on a constructive heuristic algorithm which uses *layers* of boxes that may take the shape of a set of 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 Fig. 8). The management of the feasible placement positions is based on a list of empty *maximal-spaces* as described in Lai and Chan [27]. A 3D empty space in the container is maximal if it is not contained in any 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 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, called MaxJoin, which joins maximal-spaces having the same base level. This is done so that the supporting area of the maximal-spaces is increased, increasing the likelihood that constraint *C*2 is satisfied.
(4) *Fitness evaluation*: The final phase computes the percentage volume packed, the fitness measure (or measure of quality) of the solution.

Fig. 2 illustrates the sequence of steps applied to each chromosome generated by the BRKGA.

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

4.2. Biased random-key genetic algorithm

Genetic algorithms (GA) are adaptive methods that are used to solve search and optimization problems [28,29] by associating solutions of the optimization problem with individuals of a population. Over many generations, natural populations evolve according to Charles Darwin's principle of natural selection, called *survival of the fittest* [30]. By mimicking this process, genetic algorithms, if suitably encoded, are able to *evolve* solutions to optimization 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. Goldberg [28] presents pseudo-codes for several variants of genetic algorithms.

In a GA, a solution is encoded as a set of parameters, known as *genes*, joined together to form a string of values called a *chromosome*. 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

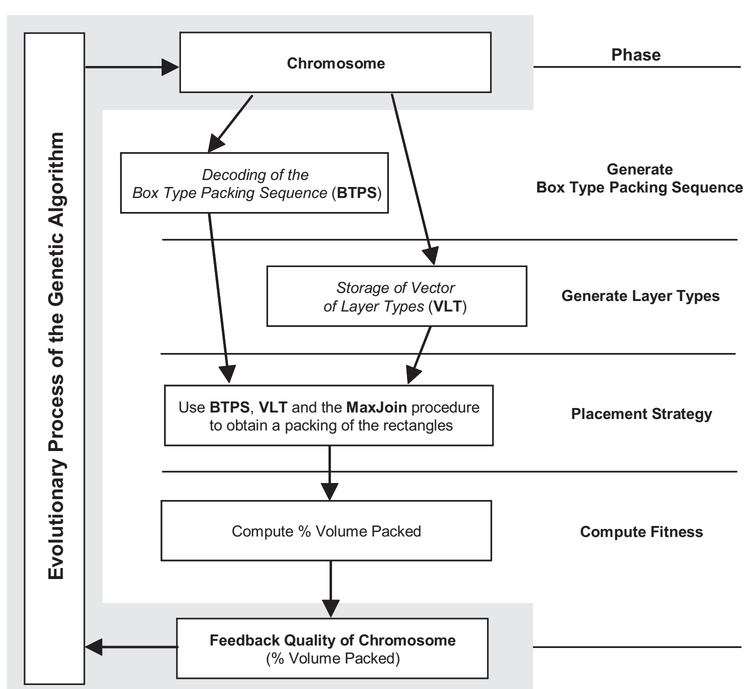

Fig. 2. Architecture of the algorithm. Evolutionary process is on the left and decoder is on the right.

reproductive phase, individuals are selected from the population and *recombined*, producing offspring, which comprise the population of 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.

4.2.1. Chromosome representation and decoding

The heuristic described in this paper is a biased random-key genetic algorithm, or BRKGA [31]. It uses a random-key alphabet comprises random real numbers between 0 and 1. The evolutionary strategy used is similar to the one proposed by Bean [32], except for the way crossover is done. An important characteristic 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 issues into the objective function evaluation. As will be shown below, 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 some 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 4.3.5.

Recall that there are K box types and that, for $k=1,\ldots,K$, 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.

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

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 procedure.

The decoding (mapping) of the first M genes of each chromosome into a *BTPS* is accomplished by sorting the genes of box types in ascending order. Fig. 3 shows an example of the decoding process for the *BTPS*. In this example there are 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 $i=1,\ldots,M$ of *VLT* is populated with $Gene_{M+i}$.

4.2.2. 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 in the literature. 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* [28]. 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.

Parametrized uniform crossover [33] 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 Fig. 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 [32], in a biased random-key genetic algorithm, we always select one parent from the set of elite solutions. Gonçalves and Resende [31] show that, compared to the random-key GA of Bean, this change produces results with better quality and converges faster to good-quality solutions. Fig. 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 5, 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

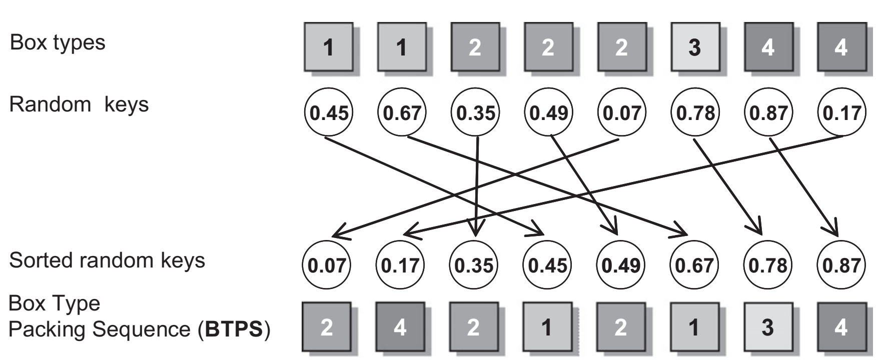

Fig. 3. Chromosome decoding procedure for the box type packing sequence.

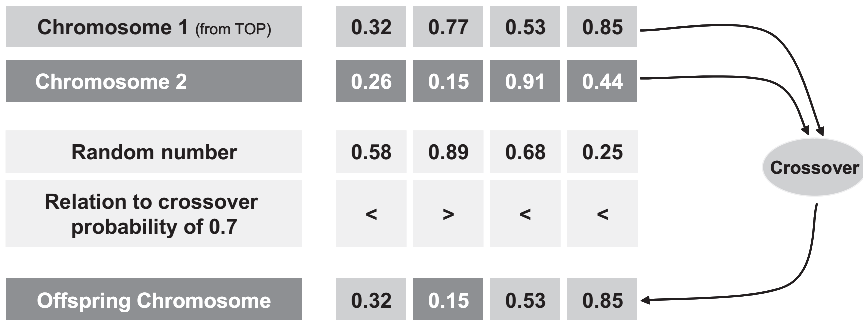

Fig. 4. Example of parametrized uniform crossover with crossover probability equal to 0.7. The offspring resembles parent 1 more than it does parent 2.

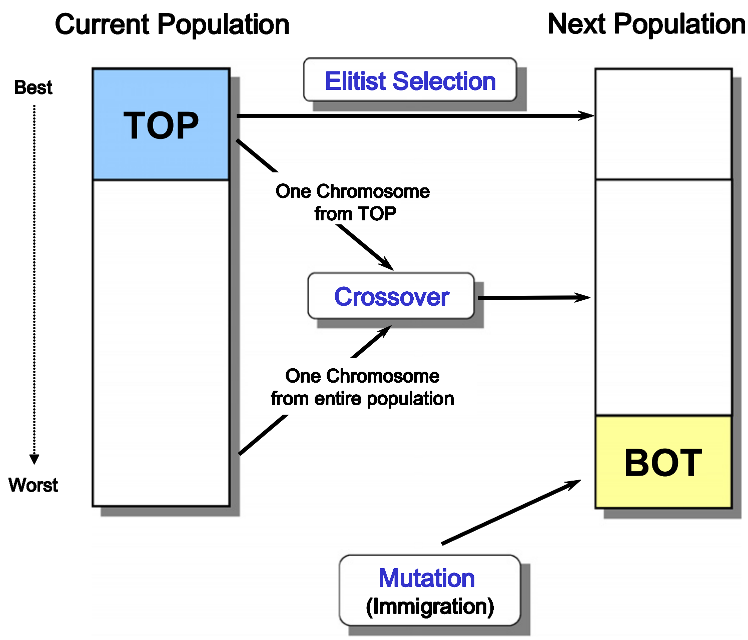

Fig. 5. Transitional process between consecutive generations.

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 Fig. 5). Like in standard mutation, the objective here is to prevent premature convergence of the population and leads to a simple statement of convergence. Fig. 5 depicts the transitional process between two consecutive generations.

4.2.3. 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

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

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

4.2.4. 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 leads 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 5, we show how this choice was determined empirically.

4.3. Placement strategy

4.3.1. 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 [27]. Fig. 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 Fig. 6a). Since the container is empty, the box is packed at the origin of the container as shown in Fig. 6b. To pack the next box, we first update the list S of empty maximal-spaces. Fig. 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.

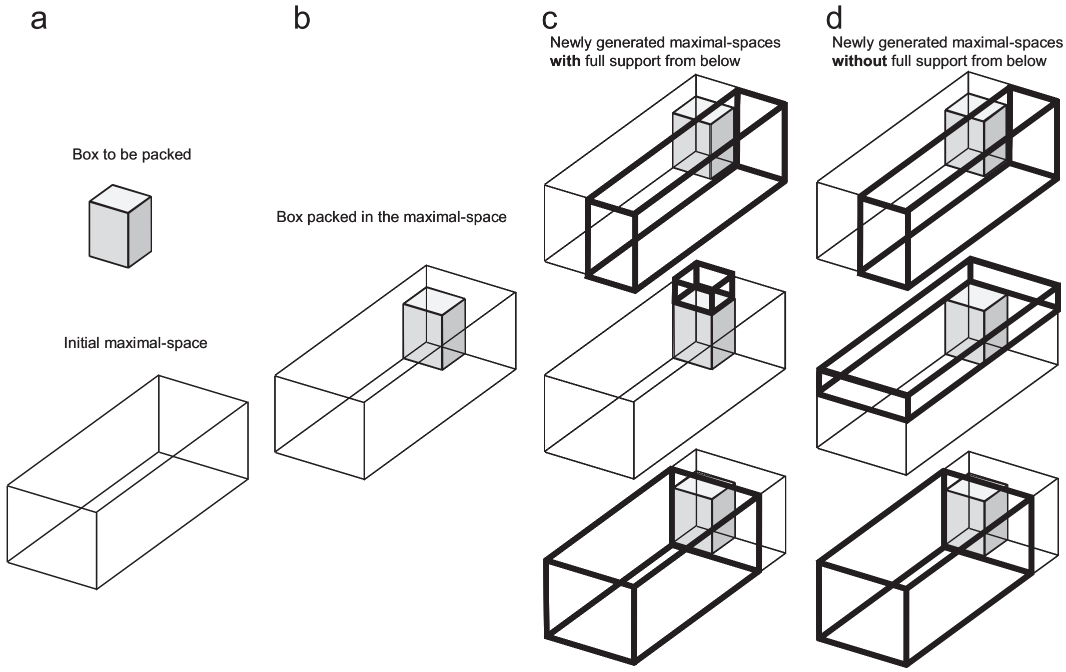

Fig. 6. Example of difference process (*DP*) with and without full support.

```
procedure BBL( b_k, S )
  Let b_k be a box of type k to be packed in the container;
  Let N_S be the number of available EMSs in S;
  Initialize X* ← L,  Y* ← W, Z* ← H;
  for i = 1, ..., N_S do
      Let x(EMS_i) be the minimum x coordinate of EMS_i;
      Let y(EMS_i) be the minimum y coordinate of EMS_i;
      Let z(EMS_i) be the minimum z coordinate of EMS_i;
      if b_k fits in EMS_i then
          if x(EMS_i) ≤ X  or (x(ERS_k) = X* and z(ERS_k) ≤ Z* )  or
             (x(ERS_k) = X* and z(ERS_k) = Z* and y(ERS_k) ≤ Y* )  then
              X* ← x(ERS_k), Z* ← z(ERS_k), Y* ← y(ERS_k) ;
              EMS* = EMS_i;
          end if
      end if
  end for
  Return EMS*;
end BBL;
```

Fig. 7. Pseudo-code of the back-bottom-left (BBL) procedure.

There are some real applications where full support from below (constraint *C*2) is not required. Fig. 6d presents the newly generated maximal-spaces generated by the *DP* procedure when full support from below is not enforced.

4.3.2. *The back-bottom-left procedure*

Recall from Section 4.3.4 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. Fig. 7 shows pseudo-code for the *BBL* procedure.

4.3.3. *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 pre-determined empty maximal-space EMS^*. Each box type can have at most six rotation variants. For each variant, we can have at most six types of layers. Therefore, we have at most 36 layer types. Fig. 8 shows all possible six layer types that can be defined for one of the six box type variants and an empty-maximal-space where the layers can be packed.

4.3.4. *Joining maximal-spaces*

The *DP* procedure presented in Section 4.3.1 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

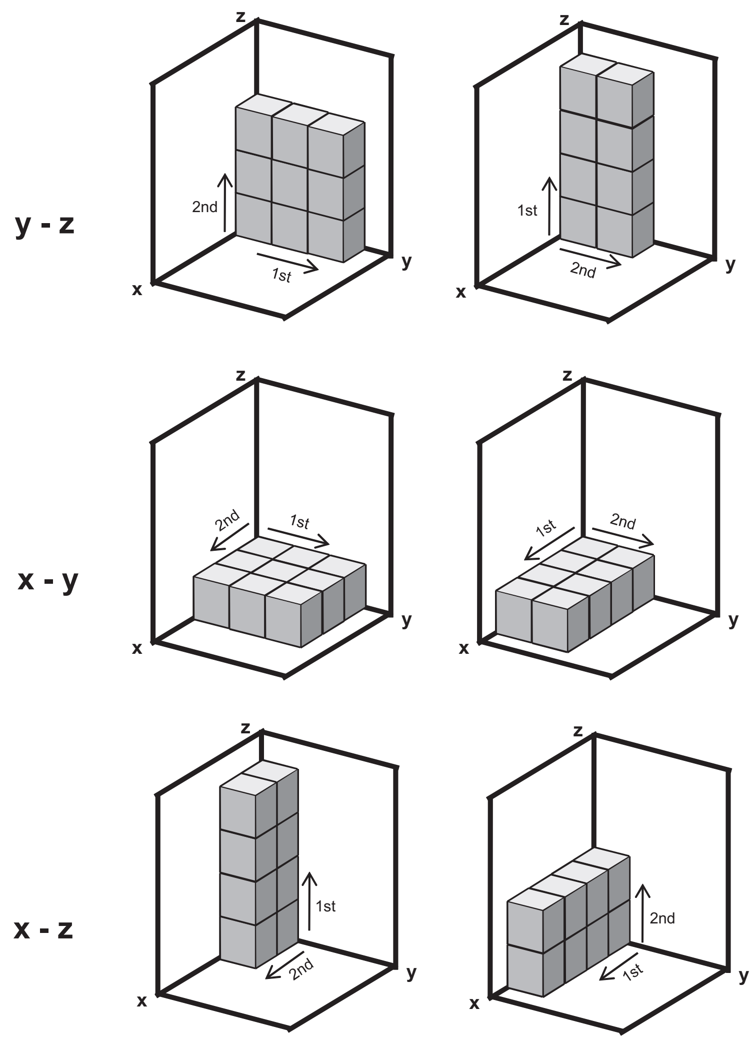

Fig. 8. Example of the six different feasible layer types for a box type rotation variant.

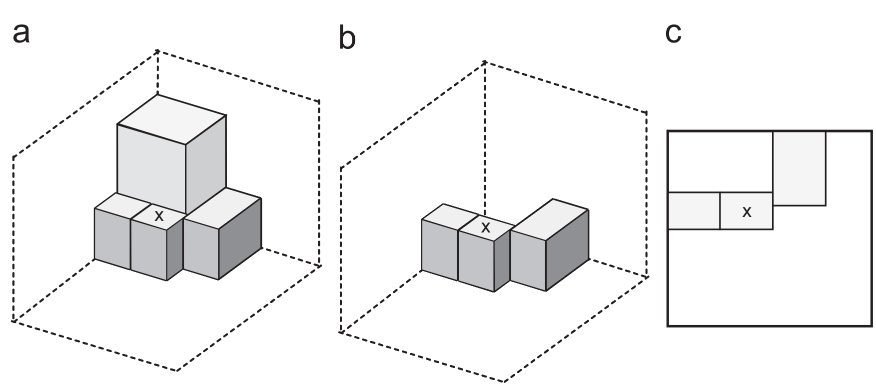

Fig. 9. Example where the empty maximal-spaces are not joined by *DP*.

does not take this into account, in such situations the possibility of satisfying constraint *C*2 (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 Fig. 9, where we assume that the box labeled x was the last one to be packed (see Fig. 9a). Fig. 9b shows all packed boxes that have the same height as box x. Fig. 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 is applied to subtract from the container the spaces corresponding to the boxes (see Fig. 10a). Note that the resulting *EMSs*, denoted by 1, 2, 3, and 4 in Fig. 10a, correspond to the complement of the *EMSs* that we seek. In the

Fig. 10. Joining spaces with the MaxJoin procedure.

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 Fig. 10b.

4.3.5. 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 composed of the following five main steps:

(1) box type selection;
(2) maximal-space selection;
(3) layer type selection;
(4) layer packing;
(5) state information update.

The pseudo-code of the placement procedure is given in Fig. 11. The box type selection step consists in choosing from *BTPS* the first box type k^* which has not yet been used (lines 9–11 of the pseudo-code). The maximal-space selection is carried out by the *BBL* procedure and the list *S* to produce EMS^* (lines 12–13 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 17–19 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 21–31 of the pseudo-code).

4.4. 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++.

5. 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.

5.1. Benchmark algorithms

We compare BRKGA-CLP with the 13 approaches listed in Table 1. These approaches are the most effective in the literature to date.

5.2. Test problem instances

The effectiveness of BRKGA-CLP is evaluated by solving the complete set of 1500 benchmark problems proposed by

```
procedure PLACEMENT (BTPS, VLT, FullSupport)
  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;
  Let S be the list of available empty EMSs;
  Let QtRemain_k be the remaining quantity of unpacked boxes of type k;
  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
  S ← Empty container;
  QtRemain_k ← N_k, Skip_k ← False, for all k;
  Placed_i ← False for all i;

  do while (There exits at least one k for which Skip_k = False);
      // ** Box type selection
      i* ← 0;
      Let i* be the first index i in BTPS for which Placed_i = False and Skip_BTPS(i) = False;
      Let k* be box type corresponding to BTPS(i*);

      // ** Maximal space selection
      EMS* ← 0;
      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;

      if EMS* = 0 then //  no EMS was found ;
          Skip_k* = False; // box type k* is not packable
      else if ;
          // ** Layer type selection
          According to QtRemain_k* fill the vector Layers
                  with all the layer-types packable into maximal-space EMS*;
          Let MaxLayers be number of layers in vector Layers;
          Let Layer* = Layers (⌈VLT(i*) × MaxLayers⌉) be the layer type selected
                  for placing the box type k*, (⌈x⌉denotes the minimum integer greater than x);

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

          // ** Information update
          Let nBox be the number of boxes of type k*contained in Layer*;
          QtRemain_k* = QtRemain_k* − nBox;
          if QtRemain_k* = 0 then Skip_k* = True; // no more boxes of type k*to pack
          Placed_i* = True;
          Update S using the DP procedure of Lai and Chan (1997);
          if FullSupport then
              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;
              Skip_k ← False for all {k | QtRemain_k > 0};
          end if
      end if
  end do
end PLACEMENT;
```

Fig. 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. [14]	Branch and bound
BG_GA	Bortfeldt and Gehring [10]	GA
BG_PGA	Bortfeldt and Gehring [43]	Parallel GA
E_TRS	Eley [15]	Tree search (TRS)
L_GH	Lim et al. [24]	Greedy heuristic
B_PTS	Bortfeldt et al. [16]	Parallel tabu search (TS)
B_NMP	Bischoff [25]	Nelder–Mead procedure
M_SATS	Mack et al. [17]	Parallel SA/TS
MO_GR	Moura and Oliveira [21]	GRASP
P_GR	Parreno et al. [22]	GRASP
P_VNS	Parreno et al. [44]	VNS
FB_TRS	Fanslau and Bortfeldt [7]	TRS
HH_HBS	He and Huang [45]	Heuristic beam search

Bischoff and Ratcliff [1] and Davies and Bischoff [2]. These instances have cargo that range from weakly heterogeneous to strongly heterogeneous. The benchmark set is divided into 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. 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.

5.3. 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 [34–41,31,42]), we obtained good results with values of TOP, BOT, and crossover probability (CProb) in the intervals shown in Table 2.

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 seeds for the random number generator) 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 the 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 problem instances. The computational results presented in the next section demonstrate that this configuration not only provides excellent results in terms of solution quality but also very robust.

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

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	15% of the next generation is made up of mutants, i.e. individuals with randomly generated chromosomes
Number of populations	3
Exchange information btw pops	Every 15 generations
Fitness	Maximize % total packed volume
Stopping criterion	Stop after 500 generations

5.4. 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 running 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 the purpose of comparison 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.

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 (200 000)	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.**

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

	Avg. time (s) for test cases BRD_01–BRD_15		
	Parreno et al. [44]	Fanslau and Bortfeldt [7]	BRKGA-CLP
Supported	–	320	232
Unsupported	238	320	147

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, which obtains overall means 01–15 which are 0.26% and 0.59% worst than those of *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 (Table 6).

6. Concluding remarks

In this paper we addressed the single container loading problem (3D-CLP), where several rectangular boxes of different sizes are to be loaded into a single rectangular container. The approach uses a novel multi-population biased random-key genetic algorithm (BRKGA). To manage the free spaces in the container we use a maximal-space representation. The proposed algorithm hybridizes a novel placement procedure with a multi-population BRKGA. 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. Two variants of the approach (with and without enforcement of full support from below) are extensively tested on the complete set of benchmark problems of Bischoff and Ratcliff [1] and Davies and Bischoff [2]. The benchmark set is made up of 1500 instances which range from weakly to strongly heterogeneous cargo. Both variants are compared with 13 other solution techniques.

The computational experiments demonstrate that not only the approach performs very well in all types of instance classes but also it obtains the best overall results when compared with other approaches published in the literature. The approach has improved the overall averages from 93.88% and 91.91% to 94.54% and 92.24%, respectively, for the unsupported and supported cases .

Acknowledgments

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

References

[1] Bischoff EE, Ratcliff MSW. Issues in the development of approaches to container loading. Omega, International Journal of Management Science 1995;23(3):377–90.
[2] Davies AP, Bischoff EE. Weight distribution considerations in container loading. European Journal of Operational Research 1999;114(3):509–27.
[3] Wäscher G, Haussner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183:1109–30.
[4] Scheithauer G. Algorithm for the container loading problem. In: Operational Research Proceedings, 1992. p. 445–52.
[5] Fekete SP, Schepers J. A new exact algorithm for general orthogonal *d*-dimensional knapsack problems. In: Algorithms—ESA '97, Lecture notes in computer science, vol. 1284. Springer; 1997. p. 144–56.
[6] Martello S, Pisinger D, Vigo D. The three-dimensional bin packing problem. Operations Research 2000:256–67.
[7] Fanslau T, Bortfeldt A. A tree search algorithm for solving the container loading problem. INFORMS Journal on Computing 2010;22(2):222–35.
[8] George AJ, Robinson DF. A heuristic for packing boxes into a container. Computers and Operations Research 1980;7(3):147–56.
[9] Loh TH, Nee AYC. A packing algorithm for hexahedral boxes. In: Proceedings of the conference of industrial automation, 1992. p. 115–26.
[10] Bortfeldt A, Gehring H. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research 2001;131(1):143–61.
[11] Pisinger D. Heuristics for the container loading problem. European Journal of Operational Research 2002;141:143–53.
[12] Gehring H, Bortfeldt A. A genetic algorithm for solving the container loading problem. International Transactions in Operational Research 1997:401–18.
[13] Bischoff EE, Janetz F, Ratcliff MSW. Loading pallets with nonidentical items. European Journal of Operational Research 1995;84:681–92.

[14] Terno J, Scheithauer G, Sommerweiss U, Riehme J. An efficient approach for the multi-pallet loading problem. European Journal of Operational Research 2000;123(2):372–81.

[15] Eley M. Solving container loading problems by block arrangement. European Journal of Operational Research 2002;141(2):393–409.

[16] Bortfeldt A, Gehring H, Mack D. A parallel tabu search algorithm for solving the container loading problem. Parallel Computing 2003;29(5):641–62.

[17] Mack D, Bortfeldt A, Gehring H. A parallel hybrid local search algorithm for the container loading problem. International Transactions in Operational Research 2004;11:511–33.

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

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

[20] Bischoff E. Three-dimensional packing of items with limited load bearing strength. European Journal of Operations Research 2004;168:952–66.

[21] Moura A, Oliveira JF. A grasp approach to the container-loading problem. IEEE Intelligent Systems 2005;20(4):50–7.

[22] Parreno F, Alvarez-Valdes R, Tamarit JM, Oliveira JF. A maximal-space algorithm for the container loading problem. INFORMS Journal on Computing 2008;20(3):412–22.

[23] Hifi M. Approximate algorithms for the container loading problem. International Transactions in Operations Research 2002;9:747–74.

[24] Lim A, Rodrigues B, Wang Y. A multi-faced buildup algorithm for three-dimensional packing problems. Omega 2003;31(6):471–81.

[25] Bischoff E. Three-dimensional packing of items with limited load bearing strength. European Journal of Operational Research 2006;168(3):952–66.

[26] Takahara S. Loading problem in multiple containers and pallets using strategic search method. In: Modeling decisions for artificial intelligence, proceedings lecture notes in artificial intelligence, vol. 3558. Berlin, Heidelberg: Springer-Verlag; 2005. p. 448–56.

[27] Lai KK, Chan JWM. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 1997;32: 115–27.

[28] Goldberg DE. Genetic algorithms in search optimization and machine learning. Addison-Wesley; 1989.

[29] Beasley D, Bull DR, Martin RR. An overview of genetic algorithms: part 1, fundamentals. University Computing 1993;15:58–69.

[30] Darwin CR. On the origin of species through natural selection. London: John Murray; 1859.

[31] Gonçalves JF, Resende MGC. Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 2010. doi: 10.1007/s10732-010-9143-1.

[32] Bean JC. Genetics and random keys for sequencing and optimization. ORSA Journal on Computing 1994;6:154–60.

[33] Spears WM, Dejong KA. On the virtues of parameterized uniform crossover. In: Proceedings of the fourth international conference on genetic algorithms, 1991. p. 230–36.

[34] Gonçalves JF, Almeida JR. A hybrid genetic algorithm for assembly line balancing. Journal of Heuristics 2002;8:629–42.

[35] Ericsson M, Resende MGC, Pardalos PM. A genetic algorithm for the weight setting problem in OSPF routing. Journal of Combinatorial Optimization 2002;6:299–333.

[36] Gonçalves JF, Resende MGC. An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering 2004;47:247–73.

[37] Gonçalves JF, Mendes JJM, Resende MGC. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 2005;167:77–95.

[38] Buriol LS, Resende MGC, Ribeiro CC, Thorup M. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks 2005;46:36–56.

[39] Buriol LS, Resende MGC, Thorup M. Survivable IP network design with OSPF routing. Networks 2007;49:51–64.

[40] Gonçalves JF. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. European Journal of Operational Research 2007;183:1212–29.

[41] Gonçalves JF, Mendes JJM, Resende MGC. A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research 2009;189:1171–90.

[42] Gonçalves JF, Resende MGC. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem, Journal of Combinatorial Optimization 2010. doi: 10.1007/s10878-009-9282-1.

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

[44] Parreno F, Alvarez-Valdes R, Oliveira JF, Tamarit JM. Neighborhood structures for the container loading problem: a VNS implementation. Journal of Heuristics, 2010. doi:10.1007/s10732-008-9081-3.

[45] He K, Huang W. Solving the single-container loading problem by a fast heuristic method. Optimization Methods and Software 2010;25(2):263–77.