A greedy adaptive search procedure for multi-dimensional multi-container packing problems

CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Greedy Adaptive Search Procedure for Multi-Dimensional Multi-Container Packing Problems

Teodor Gabriel Crainic
Guido Perboli
Roberto Tadei

March 2012

CIRRELT-2012-10

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Greedy Adaptive Search Procedure for Multi-Dimensional Multi-Container Packing Problems

Teodor Gabriel Crainic[1,2,*], Guido Perboli[1,3], Roberto Tadei[3]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Management and Technology, Université du Québec à Montréal, P.O. Box 8888, Station Centre-Ville, Montréal, Canada H3C 3P8

[3] Department of Control and Computer Engineering, Politecnicò di Torino, Corso Duca degli Abruzzi, 24 - I-10129 Torino, Italy

Abstract. Multi-dimensional multi-container packing problems appear within many settings of theoretical and practical interest, including Knapsack, Strip Packing, Container Loading, and Scheduling problems. These various problem settings display different objective functions and constraints, which may explain the lack of efficient heuristics able to jointly address them. In this paper we introduce GASP - Greedy Adaptive Search Procedure, a metaheuristics able to efficiently address two - and three-dimensional multi-container packing problems. GASP combines the simplicity of greedy algorithms with learning mechanisms, aiming to guide the overall method towards good solutions. Extensive experiments indicate that GASP attains near-optimal solutions in very short computing times. GASP also improves state-of-the-art results, when using the same computing times.

Keywords. Multi-Dimensional Bin Packing, Greedy Adaptive Search.

Acknowledgements. While working on this project, T.G. Crainic was the Natural Sciences and Engineering Research Council of Canada (NSERC) Industrial Research Chair in Logistics Management, ESG UQAM, and Adjunct Professor with the Department of Computer Science and Operations Research, Université de Montréal, and the Department of Economics and Business Administration, Molde University College, Norway. This project was partially funded by the Ministero dell'Istruzione, Università e Ricerca (MIUR) (Italian Ministry of Education, University, and Research), under the Progetto di Ricerca di Interesse Nazionale (PRIN) 2009, "Methods and Algorithms for the Logistics Optimization", NSERC, through its Industrial Research Chair and Discovery Grant programs, and by the partners of the Chair, CN, Rona, Alimentation Couche-Tard, and the Ministry of Transportation of Québec.

---

* Corresponding author: TeodorGabriel.Crainic@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2012

1 Introduction

Packing problems have been much studied in the past decades due, in particular, to their wide range of applications in many settings of theoretical and practical interest, e.g. packing, scheduling, and routing. Packing problems all have an identical structure, recently summarized in the classification of Wäscher et al. (2007). Let us define two sets of elements in one or more (usually two or three) geometric dimensions: a set of large items, often called *bins*, knapsacks or containers, and a set of small items, usually referred as *items*. The goal is to select all or some of the items, group them into one or more subsets, and assign each of these subsets to one of the bins such that the geometric condition holds. This means that the items of each subset lie entirely within the bin without overlapping.

Multi-dimensional multi-container packing problems play a central role in planning freight and supply chain systems in order to reduce costs and improve the use of facilities and equipment. These problems are encountered as support to operational decisions or as part of more complex tactical decision processes. Despite its importance, research on multi-dimensional multi-container packing problems is relatively recent (Martello et al., 2000), and state-of-the-art methods require high computing times for relatively small instances (Crainic et al., 2009). Therefore, there is a need for accurate and fast solution methods able to deal with larger instances. This is the goal of this paper. A related issue concerns the development of industrial-strength codes implementing state-of-the-art algorithms. Actually it is often the case that several methods address the same problem, but that each performs better on particular problem instances. For example, up to the work by Crainic et al. (2008), at least four heuristics were competing to solve the 2D and 3D Bin Packing Problem, but none proved to be really dominant. An additional difficulty comes from the relatively high number of specific parameters one must tune for each heuristics targeting a particular packing problem class.

In this paper, we introduce *GASP - Greedy Adaptive Search Procedure*, a new framework for multi-dimensional multi-container packing problems. *GASP* combines the simplicity of greedy algorithms with learning mechanisms which aim to guide the overall method towards good solutions.

GASP was designed according to the following principles:

- efficiency: achieving near-optimal solutions with limited computing effort. Indeed, existing methods need minutes or hours to solve instances with hundreds of items (Crainic et al., 2009; Hadjiconstantinou and Iori, 2007; Bortfeldt, 2006), whilst much larger problem instances (e.g., 4000 items) may be found in actual supply chain and production applications;

- simplicity: a modular and easy-to-understand algorithmic structure with a limited set of parameters.

The main idea underlying GASP is to separate the feasibility phase, addressing the item packing, from the optimality phase. This is accomplished by using a reactive, problem-specific *scoring* phase to order the items, which are then packed by an *on-line constructive* heuristics common to all problem classes considered. The use of a unique item-placing heuristics handles possible additional packing constraints that can arise in practical situations. Moreover, the scoring phase incorporates problem-specific knowledge and provides the means to implicitly explore the solution space without an explicit neighborhood exploration, which could be computationally expensive due to item accommodation in multi-dimensional bins.

Extensive experiments on 2D and 3D multi-container packing problems indicate that *GASP* attains near-optimal solutions in very short computing times, and improves state-of-the-art results, when using the same computing times.

The paper is organized as follows: Section 2 describes in detail Bin Packing problems (which formulate multi-container problems) in their two and three-dimensional variants and gives a survey of the related literature. Section 3 introduces GASP, whose computational results are shown in Section 4. Finally, in Section 5 we summarize our results.

2 Problem statement and literature review

In multi-dimensional multi-container packing problems, given a set of multi-dimensional items and a set of homogeneous containers, our aim is to load all items into the minimal number of containers. There are many different practical situations within this context, where the firm already owns its containers or their acquisition or leasing costs are already accounted for.

The problem can be formulated as a Multi-Dimensional (in particular, Three-Dimensional and Two-Dimensional) Bin Packing Problem. Formally, given a set of box items $i \in I$, with sizes w_i, l_i, and h_i, and an unlimited number of bins of fixed sizes W, L, and H, the *Three-Dimensional orthogonal Bin Packing Problem* (*3D-BPP*) consists in orthogonally packing items into the minimum number of bins. According to the typology introduced in Wäscher et al. (2007), the problem is also known as the *Three-Dimensional Single Bin-Size Bin Packing Problem* (*3D-SBSBPP*).

In several freight applications, items cannot be piled. This is the case, for example, with the transport of furniture. The problem then reduces to the *Two-*

Dimensional orthogonal Bin Packing Problem (*2D-BPP* or *2D-SBSBPP*).

The method we propose addresses both problems. In the following we briefly describe some of the features of the two problems. For a more detailed description, the reader can refer to the recent survey by Crainic et al. (forthcoming).

2D-BPP was the first Multi-Dimensional Bin Packing variant to be studied. The literature on this topic is extensive. Thus in the following we recall only the main results. $TSPACK$ is the tabu search algorithm for the *2D-BPP* developed by Lodi et al. (1999). This algorithm uses two simple constructive heuristics to pack items into bins. The tabu search only controls the movement of items between bins. Two neighborhoods are considered to try to relocate an item from the weakest bin (i.e., the bin that appears to be the easiest to empty) into another. Since the constructive heuristics produces guillotine packings, so does the overall algorithm. The algorithm is presently the best metaheuristics for the *2D-BPP*, but it requires a computing effort in the order of 60 CPU seconds per instance.

The same authors presented a shelf-based heuristics for the *2D-BPP*, called *Height first - Area second* (*HA*) (Lodi et al., 2004). The algorithm chooses the best of two solutions. To obtain the first one, items are partitioned into clusters according to their height and a series of layers are obtained from each cluster. The layers are then packed into the bins by using the Branch-and-Bound approach for the *1D-BPP* problem by Martello and Toth (1990). The second solution is obtained by ordering the items by non-increasing area of their base and bulding new layers. As previously, the layers are packed into the bins by solving a *1D-BPP* problem. The method is faster, but less accurate than $TSPACK$.

The *3D-BPP* was next introduced in the literature. The first exact method for the *3D-BPP* was a two-level Branch-and-Bound proposed by Martello et al. (2000). The first level assigns items to bins. At each node of the first-level tree, a second level Branch-and-Bound is used to verify whether the items assigned to each bin can be packed into it. In the same paper, the authors introduced two constructive heuristics. The first, called *S-Pack*, is based on a layer-building principle derived from the shelf approach. The second, called *MPV-BS*, repeatedly fills one bin after the other by means of the Branch-and-Bound algorithm for the single container presented by the authors in the same paper. A computing time of 1000 CPU seconds was imposed.

Faroe et al. (2003) presented a Guided Local Search (GLS) algorithm for the *3D-BPP*. Starting with an upper bound on the number of bins obtained by a greedy heuristics, the algorithm iteratively decreases the number of bins, each time using GLS to search for feasible packing. The process terminates when a given time limit has been reached or the upper bound matches a precomputed lower bound.

Computational experiments were reported for 2D and 3D instances with up to 200 items. The results were satisfactory, but required a computing time in the order of 1000 CPU seconds.

Crainic et al. (2008) defined the Extreme Points (EPs) rule to identify possible positions to place items into a given (partially loaded) bin. The EPs extend the Corner Points of Martello et al. (2000) to better exploit the bin volumes. They are independent of the particular packing problem addressed and can easily handle additional constraints, such as fixing the item position. EPs were introduced into the well-known Best First Decreasing (BFD) heuristics, producing the *EP-BFD* heuristics for the *1D-BPP*. Extending the *EP-BFD* to the *3D-BPP* proved far from trivial however, as the item ordering in higher dimensions may be affected by more than one attribute (e.g., volume, side area, width, length, and height of the items). Several sorting rules were then tested and the best ones were combined into the *C-EPBFD*, a composite heuristics based on *EP-BFD*. Extensive experimental results showed that the *C-EPBFD* requires a negligible computing effort and outperforms both current constructive heuristics for the *3D-BPP* and more complex methods, e.g. the truncated Branch-and-Bound by Martello et al. (2000).

Crainic et al. (2009) proposed TS^2PACK, a two-level tabu search metaheuristics for the *3D-BPP*. The first level is a tabu search method that changes the assignment of items to bins. For each assignment, the items assigned to a bin are packed by means of the second-level tabu search, which uses the Interval Graph representation of the packing by Fekete and Schepers (2004) to reduce the search space. The accuracy of the metaheuristics is enhanced by the k-chain-move procedure, which increases the size of the neighborhoods without increasing the overall complexity of the algorithm. TS^2PACK currently obtains the best solutions for the *3D-BPP*. Nevertheless, the method has a rather slow convergence rate, requiring 300 CPU seconds to find the best solution.

Literature reviews emphasize that the state-of-the-art methods lack in either accuracy or efficiency (Crainic et al., forthcoming). The method we propose aims to address this challenge.

3 The GASP framework

The main idea of the method we propose is to separate how items are packed, the *feasibility* phase, from the selection of the packing order of the items, the *optimality* phase.

We deal with the feasibility phase by means of a greedy procedure. In the

optimality phase, the order of the items to be packed is determined by *scores*, which represent the relative value of an item. We embed these elements into a metaheuristic framework, which provides a learning mechanism used to update the scores.

In more detail, we assign a *score* to each item, thus specifying the order in which items are to be considered by the accommodation heuristics. The score computation is problem specific and incorporates knowledge regarding how item and bin attributes (e.g., dimensions) interact, the problem objectives, etc. For some packing problems, the particular characteristics of the problem instances may also influence the desired order. This is reflected in GASP through an update mechanism, which modifies the scores at each iteration based on the performance of previous iterations.

Scores are first initialized by the *Score Initialization* procedure, and then they are dynamically modified by means of the *Score Update* and the *Long-term Score Reinitialization* procedures. *Score Update* proceeds through small changes, aiming to adjust the scores used to sort items at iteration k of GASP according to the quality of the solution built at iteration $k-1$. *Long-term Score Reinitialization* incorporates long-term decisions, as long-term memory structures, and proceeds through larger score modifications in order to avoid cycling on the same solutions and to explore new regions of the solution space.

Score computation and updates depend upon a number of parameters. We aim to keep this number as small as possible and simplify their adjustment during computation. If needed, GASP provides a problem-specific, dynamically-adjusting parameter procedure denoted *Parameter Update*.

The main steps of *GASP* can be summarized as follows (see Fig. 1):

- Build an initial solution by means of the *C-EPBFD* and set it as the best solution BS;
- Scoring Phase:
  - Initialize the scores (*Score Initialization* procedure)
  - While *Stopping Conditions* are not encountered, repeat the following steps:
    - Sort the items by non-increasing score and apply the greedy procedure (*Greedy)*, obtaining a new current solution CS
    - If a given number of successive non-improving iterations is reached, reinitialize the scoring using the *Long-term Score Reinitialization* procedure; otherwise, update the scores using the *Score Update* procedure applied to CS
    - If CS is better than BS, then set BS to CS

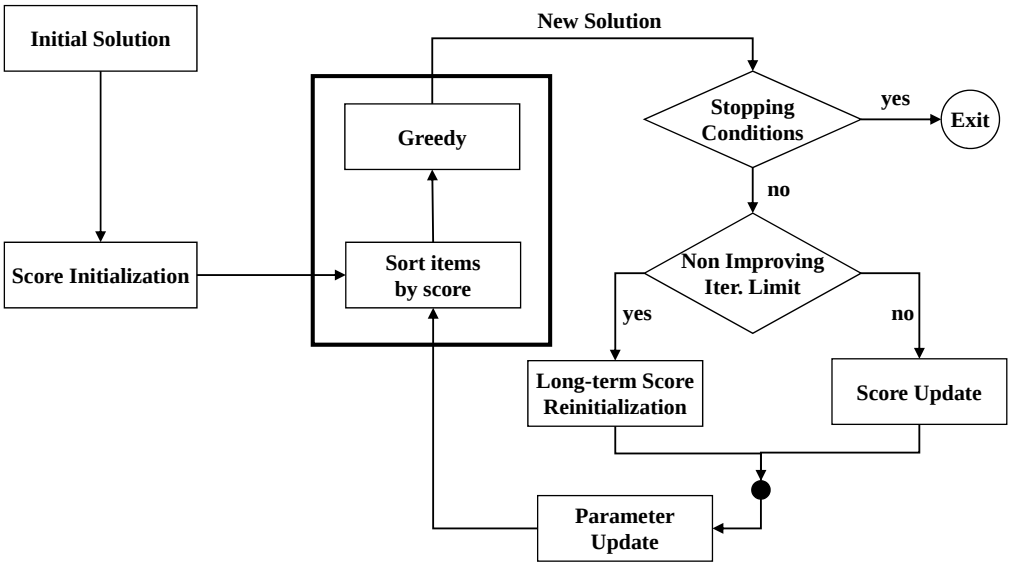

Figure 1: General scheme of *GASP*

We now detail the various steps of *GASP*.

3.1 Initial Solution

The algorithm is initialized by means of $C - EPFFD$ by Crainic et al. (2008) because of its good performance both in 2D and 3D problems.

3.2 Score Initialization

Given the item ordering associated to the initial solution, the initial scores take values from n to 1, where n is the number of items. More precisely, the score of the first item of the list is set to n, that of the second one to $n - 1$, and so on.

3.3 Greedy

The procedure is based on the *Best Fit Decreasing* (BFD) idea and generalizes the heuristics presented in Crainic et al. (2008). Following an initial item sorting by non-increasing volume, the BFD constructive heuristics for 1D Bin Packing problem tries to load each item into the best bin. This bin is defined as the bin which, after loading the item, has the maximum free volume, given by the bin volume minus the total volume of the items it contains. A new bin is created whether the item cannot be loaded into the existing bins. Despite its simplicity, the BFD heuristics offers good performances for 1D Bin Packing problems. Similar heuristics exist for other packing problems, e.g., Knapsack and Strip Packing. Unfortunately, extending these heuristics to a general constructive heuristics for multi-dimensional problems is not a trivial task. On the one hand, while in 1D cases the ordering is done considering a unique attribute characterizing both items and bins (i.e. their volume or profit), more choices exist in the multi-dimensional context. One may thus consider sorting items according to their width, height, depth, their volume or the areas of their different faces. Consequently, the definition of the best bin in the BFD heuristics is not unique. While the item placement does not need to be considered in 1D problems, a 2D or 3D packing may vary significantly according to how items are placed inside the bin, even when the item ordering and the rule selecting the best bin are not changed. Moreover, according to the packing problem, the number of available bins may be unlimited or fixed and all items or just a subset of them must be loaded.

We propose a constructive heuristics based on BFD ideas, denoted *Extreme-Point Best Positioning Heuristics* (*EP-BFD*), which places the items into bins by using the Extreme Point concept (Crainic et al., 2008). The Extreme Points (*EP*s) define the points where one may place an item to be added to an existing packing. Assume that item k with width, length, and depth w_k, l_k, and h_k, respectively, is already placed in position (x_k, y_k, z_k) of a bin. A new item j can then be placed at specific points (the *EP*s) which are the orthogonal projections of the points $(x_k + w_k, y_k, z_k)$, $(x_k, y_k + l_k, z_k)$, and $(x_k, y_k, z_k + h_k)$ on the three axes (see Figure 2).

The main steps of the *EP-BFD* (see Algorithm 1) are as follows:

- Order the items by non-increasing score;
- For each item in the resulting sequence, find the best *EP* of the best available bin into which the item will be loaded;
- If such a bin exists, load the item into it on the given *EP*;
- Otherwise either a new bin is created if the total number of bins does not

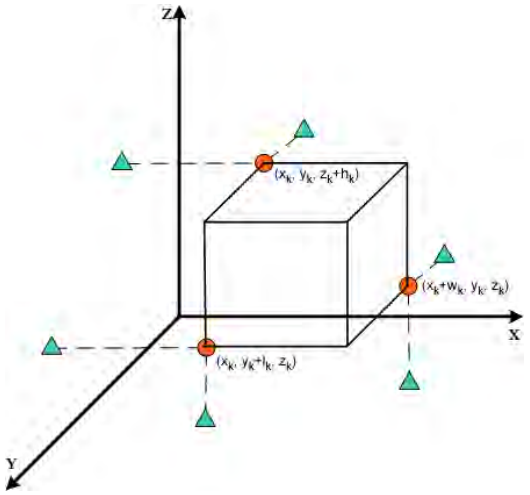

Figure 2: Definition of Extreme Points in 3D Packing

exceed the given maximum or the item is discarded.

Changing the maximum number of available bins adapts *EP-BFD* to different packing problems. For example, the number of bins is infinite for Bin Packing problems, while it is equal to 1 for Knapsack, Strip Packing, and Container Packing problems. Moreover, in specific logistics applications restrictions on the available bins can be present, as, for example, in the Variable Sized and Cost Bin Packing problem (Crainic et al., 2011). The behavior of *EP-BFD* depends on how the best *EP*s are selected. Computational experiments have shown that the best trade-off between accuracy and efficiency is given by the *Residual Space* (*RS*) rule (Crainic et al., 2008). The *RS* measures the free space available around an *EP*. More precisely, when an *EP* is created, its *Residual Space* on each axis is set equal to the distance between its position and the side of the bin along that axis (Figure 3a). Given item k to be loaded, the algorithm puts this item on the *EP* that minimizes the difference between its own *RS* and the item size:

$$min_{e \in \{EP\}} = [(RS_e^x - w_k) + (RS_e^y - d_k) + (RS_e^z - h_k)] \quad (1)$$

where RS_e^x, RS_e^y, and RS_e^z are the *RS*s of *EP* e on X, Y, and Z axes, respectively. Every time an item is added to the packing, all the *EP*s and their *RS*s are updated. Figure 3b illustrates the concept. For complex packings, the *RS* gives only an estimate of the effective volume available around an *EP* and, thus, potential overlaps with other items have to be checked when packing a new item on a chosen *EP* (see Crainic et al. (2008) for further details).

Algorithm 1 Extreme-Point Best Positioning Heuristics

Input $whatIsBest$: Type of best criterion
Input I : List of items to be loaded
Input B_{max} : Maximum number of available bins

$canLoad(i, b)$: Function that tries to load item i into bin b on each possible extreme point and then it returns the best point found (and its relative merit value).

Sort items by non-increasing score.

for all $i \in I$ **do**
  bool $isLoadable$ = **false**
  int $bestResult$ = 0
  for all $b \in BINLIST$ **do**
    if ((merit,resPoint) = canLoad(i, b)) **then**
      if ($merit > bestResult$) **then**
        $isLoadable$ = **true**
        $bestPoint$ = $resPoint$
        $bestBin$ = b
        $bestResult$ = $merit$
      end if
    end if
  end for
  if (isLoadable == **true**) **then**
    LoadItPoint(i, $bestBin$, $bestPoint$)
  else
    if $|BINLIST| < B_{max}|$ **then**
      Create $newBin$
      LoadIt(i, $newBin$)
      $BINLIST = BINLIST \cup \{newBin\}$
    else
      Discard item i
    end if
  end if
end for

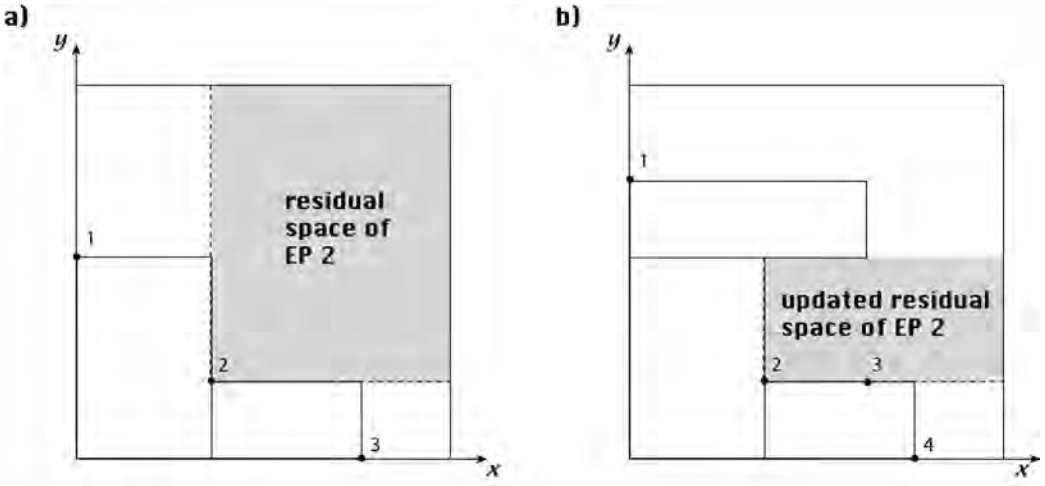

Figure 3: Example of *Residual Space*

3.4 Long-term Score Reinitialization

Similar to the *Score Initialization* procedure, but starting from the item list of the best solution found so far. The first item of the list has its score value s_i set to n, the second one to $n-1$, and so on.

3.5 Score Update

Bin Packing constructive heuristics generally yield very good packings for the first bins and rather poor ones for the last ones. Moreover, "mistakes" in the item ordering are usually to be found in the central portion of the item list and involve a relatively small number of items that should be swapped. But, of course, these items are not known a-priori. The main idea is then to try to force item swaps between bins that are considered "well-packed" and the others, by modifying the scores according to the following rule:

$$s_i = \begin{cases} s_i\,(1-m) & b(i) \in \mathcal{B}' \subset \mathcal{B} \\ s_i\,(1+m) & \text{otherwise} \end{cases} \tag{2}$$

where m is a positive parameter to be calibrated, $b(i)$ is the bin where item i is loaded, and $\mathcal{B}'$ is the subset of the loaded bins $\mathcal{B}$ that are considered well-packed.

The rule (2) penalizes the items loaded into the well-packed bins and helps, by increasing their score, the items loaded into the other ones. Consider the order

of bins in the set $\mathcal{B}$ as defined by the sequence from 1 to $|\mathcal{B}|$ generated by the bin creation (i.e., the first time an item is allocated to a bin). Then, according to our tests, bins in the first half of this sequence may be considered well-packed, i.e., $\mathcal{B}' = \left\{1, \ldots, \lfloor \frac{|\mathcal{B}|}{2} \rfloor \right\}$.

Obviously, the value of m strongly affects the behavior of (2), the score modification being directly proportional to m. Thus, the larger the value of m, the higher the number of potential "swaps" and the more potentially diverse a solution is when compared to that of the previous iteration. This solution-diversifying behavior may be counter-productive, however, when the goal is to refine the search around a solution by finding the right sorting of just a few items. A smaller value of m, intensifying the search by producing smaller changes in the item scores, would then be appropriate. The goal then is to use different values of m at various stages of the search, values that may self-adjust according to the instance data and the search trajectory of the heuristics.

Notice that the amplitude of the score modification (i.e. the value of m) may be changed either by varying the percentage of modification of the previous score or by increasing/decreasing the number of possible changes in the item sorting. We then propose a *Score Update* mechanism, which proceeds along a two-stage direction. The first stage starts with the largest value of m and gradually decreases it by reducing the maximum percentage of score variation. For each maximum score-variation value, the second stage gradually reduces the number of items that can be swapped in the list. This is implemented by making m dependent upon two positive parameters:

- p, which affects the maximum percentage of the score modification. The value of p is initially set to 1 and is modified following each *Long-term Score Reinitialization*. The initial maximum percentage of the score modification, $\overline{s}$, is experimentally set at 10%;
- k, the number of possible item swaps. The value of k is set initially to 1 (and reset to 1 after each *Long-term Score Reinitialization*) and is increased according to the search trajectory (each time the best solution is updated) by the *Parameter Update* mechanism. Its maximum value, k_{max}, is experimentally set to 4.

We therefore introduce a parametric definition of m

$$m = \frac{\overline{s}}{p}(k_{max} - k), \tag{3}$$

yielding the following expression for the score updating rule

$$s_i = \begin{cases} s_i \left(1 - \frac{\bar{s}}{p}(k_{max} - k)\right) & b(i) \in \left\{1, .., \lfloor \frac{p}{2} \rfloor\right\} \\ s_i \left(1 + \frac{\bar{s}}{p}(k_{max} - k\right) & \text{otherwise} \end{cases} \tag{4}$$

which provides *GASP* with the desired capability of making both large diversification and more precise intensification score modifications as appropriate.

3.6 Parameter Update

The two parameters are dynamically updated:

- p is increased by 1 after each *Long-term Score Reinitialization*
- k is increased by 1 every time the best solution is updated and set to 1 after each *Long-term Score Reinitialization*.

4 COMPUTATIONAL RESULTS

Experiments were carried out on standard benchmark instances. For *2D-BPP*, we considered ten classes of instances from Berkey and Wang (1987) (Classes I-VI) and Martello and Vigo (1998) (Classes VII-X). For each class, we considered instances with a number of items equal to 20, 40, 60, 80, and 100. For each class and instance size, 10 instances were generated (the code of the generator and the instances are available at *http://www.or.deis.unibo.it/research.html*).

Martello et al. (2000) proposed seven classes of instances for the *3D-BPP*. For Classes *I* to *V*, the bin size was $W = H = D = 100$ and the items were of five types, ranging from small to large-sized items. The five classes mixed the item types in order to test different usage scenarios. In classes *VI* to *VIII* bin and item dimensions varied according to the following rules:

- *Class VI*: w_i, l_i, $h_i \sim$ U[1,10] and $W = L = H = 10$ (where U is the uniform distribution);
- *Class VII*: w_i, l_i, $h_i \sim$ U[1,35] and $W = L = H = 40$;
- *Class VIII*: w_i, l_i, $h_i \sim$ U[1,100] and $W = L = H = 100$.

Following Martello et al. (2000), Faroe et al. (2003), and Crainic et al. (2009), we did not consider Classes II and III, which displayed properties similar to those of Class I. For each remaining class, i.e., I and IV to VIII, we considered instances with 50, 100, 150, and 200 items. Given a class and an instance size, we generated 10 different problem instances based on different random seeds.

GASP was coded in C++ and runs were performed on a Pentium4 3 GhZ workstation. Parameter values were determined through a tuning phase performed on a subset of twenty 2D and 3D instances. The parameter values were set as follows: The time limit was 3 and 5 seconds for 2D and 3D problems respectively, $\overline{s} = 0.1$, $k_{max} = 4$, $k = 1$, and $p = 1$. The *Long-term Score Reinitialization* procedure was applied every 1000 iterations.

4.1 2D-BPP results

We compare the results of *GASP* to those of $TSPACK$, the tabu search of Lodi et al. (1999), as well as to the best results from the literature obtained by heuristic and exact approaches. $TSPACK$ was coded in C and run on a Silicon Graphics INDY R10000sc (195 MHz) with a time limit of 60 CPU seconds for each instance Lodi et al. (1999). A 3-second time limit is given to *GASP*.

The results are summarized in Table 1. The instance type is given in the first column, while Columns 2, 3, and 4 present the results of *GASP*, $TSPACK$, and the best known solution taken from the literature (the optimal value in most cases), respectively. Notice that the best known solutions have generally been obtained by means of different exact methods and with a computing effort of several thousands of seconds. Finally, Columns 5 and 6 give the relative percentage gaps of *GASP* with respect to $TSPACK$ and the best known solutions (a negative value means a better performance of *GASP*).

GASP achieves better results than $TSPACK$, while reducing the computing effort by more than one order of magnitude. Moreover, *GASP* reaches results that are lower than 1% of the best known results. We stress that these results are often the optima obtained by a significant computing effort.

4.2 3D-BPP results

For the 3D case, *GASP* is compared to GLS (Faroe et al. (2003)), MPV, the truncated Branch-and-Bound proposed in Martello et al. (2000), and TS^2PACK (Crainic et al. (2009)).

GLS was coded in C and results were obtained with a time limit of 1000 CPU seconds for each instance on a Digital workstation with a 500 MHz CPU. Algorithms MPV and TS^2PACK were coded in C and run on a Pentium4 with 2000 MHz CPU. A time limit of 1000 CPU seconds per instance was imposed to MPV. The limit was 300 CPU seconds for TS^2PACK, equivalent to 1000 CPU seconds for the Digital 500 workstation (according to the SPEC CPU2006

Table 1: *2D-BPP*: Comparison of *GASP* and State-of-the-Art Methods

Class	GASP 3 ss	TSPACK 60 ss	UB*	Gap TSPACK	Gap UB*
I	100.1	101.5	99.7	-1.40%	0.40%
II	12.9	13	12.4	-0.81%	4.03%
III	70.6	72.3	68.6	-2.48%	2.92%
IV	13	12.6	12.4	3.23%	4.84%
V	90.1	91.3	89.1	-1.35%	1.12%
VI	11.8	11.5	11.2	2.68%	5.36%
VII	83.1	84	82.7	-1.09%	0.48%
VIII	83.6	84.4	83	-0.96%	0.72%
IX	213	213.1	213	-0.05%	0.00%
X	51.4	51.8	50.4	-0.79%	1.98%
Total	**729.6**	**735.5**	**722.5**	**-0.82%**	**0.98%**

benchmarks Standard Performance Evaluation Corporation (2006)). A time limit of 5 CPU seconds per 3D problem is allocated to *GASP*, to better represent circumstances when 300 second computing times are not acceptable.

Table 2 displays performance measures comparing *GASP* to the state-of-the-art algorithms. Column 1 gives the instance type, bin dimension, and number of items. Column 2 presents the results of *GASP*, while Columns 3-6 give the gaps of the solutions obtained by *GASP* relative to those of MPV, GLS, TS^2PACK, and L_B, respectively. The gaps were computed as $(mean_{GASP} - mean_o)/mean_o$, where, for a given set of problem instances, $mean_{GASP}$ and $mean_o$ are the mean values obtained by *GASP* and the compared method respectively. A negative value means that *GASP* yields a better mean value. The last row displays the total number of bins used computed as the sum of the values in the column, and the average of the mean gaps.

The results indicate that *GASP* performs better than the truncated Branch-and-Bound and has a gap of only 0.9% with the best algorithm in the literature, with a negligible computing time: 5 CPU seconds compared to 1000 for GLS and 300 for TS^2PACK.

To further illustrate this efficiency, Table 3 displays the performance of *GASP* w.r.t. those of GLS and TS^2PACK, in comparable computing times (i.e., 60 CPU seconds for GLS, which runs on a Digital 500 workstation, and 18 seconds for TS^2PACK, which runs on a Pentium4 2000, Standard Performance Evaluation Corporation (2006)). These results show that *GASP* actually improves the solutions of both GLS and TS^2PACK up to 0.6% on average.

Table 2: *3D-BPP*: Comparison of *GASP* and State-of-the-Art Methods

Class	Bins	n	GASP 5 ss	MPV 1000 sec	GLS 1000 sec	TS^2PACK 1000 sec	LB
I	100	50	13.4	-1.47%	0.00%	0.00%	3.88%
		100	26.9	-1.47%	0.75%	0.75%	5.08%
		150	37	-3.14%	0.00%	0.00%	3.35%
		200	51.6	-1.34%	0.78%	0.98%	3.82%
IV	100	50	29.4	0.00%	0.00%	0.00%	1.38%
		100	59	-0.17%	0.00%	0.17%	0.85%
		150	86.8	-0.46%	0.00%	0.00%	0.46%
		200	118.8	-0.59%	-0.17%	0.00%	0.42%
V	100	50	8.4	-8.70%	1.20%	1.20%	10.53%
		100	15.1	-13.71%	0.00%	-0.66%	7.86%
		150	20.6	-14.17%	1.98%	2.49%	9.57%
		200	27.7	-12.89%	1.84%	1.09%	6.54%
VI	10	50	9.9	1.02%	1.02%	1.02%	5.32%
		100	19.1	-1.55%	0.00%	0.00%	3.80%
		150	29.5	-0.34%	0.34%	1.03%	3.51%
		200	38	-0.52%	0.80%	0.80%	3.54%
VII	40	50	7.5	-8.54%	1.35%	1.35%	10.29%
		100	12.7	-16.99%	3.25%	3.25%	10.43%
		150	16.6	-15.74%	5.06%	5.06%	15.28%
		200	24.2	-13.88%	2.98%	2.98%	6.61%
VIII	100	50	9.3	-7.92%	1.09%	1.09%	6.90%
		100	19	-5.94%	0.53%	1.06%	3.26%
		150	24.8	-9.16%	3.77%	3.77%	10.22%
		200	31.1	-10.89%	4.01%	3.67%	10.28%
Total			**736.4**	**-4.35%**	**0.85%**	**0.90%**	**3.89%**

Table 3: *3D-BPP*: Comparison of *GASP* and State-of-the-Art Methods when using the same computing times

Class	Bins	n	GASP 5 ss	GLS 60 ss	TS^2PACK 18 ss	LB
I	100	50	13.4	0.00%	0.00%	3.88%
		100	26.9	0.00%	-0.37%	5.08%
		150	37	-1.33%	-1.86%	3.35%
		200	51.6	-2.27%	-2.64%	3.82%
IV	100	50	29.4	0.00%	0.00%	1.38%
		100	59	0.00%	-0.34%	0.85%
		150	86.8	-0.34%	-0.57%	0.46%
		200	118.8	-0.92%	-0.34%	0.42%
V	100	50	8.4	1.20%	1.20%	10.53%
		100	15.1	0.00%	-1.95%	7.86%
		150	20.6	-0.48%	-1.44%	9.57%
		200	27.7	-0.36%	-1.07%	6.54%
VI	10	50	9.9	1.02%	0.00%	5.32%
		100	19.1	-1.04%	-2.05%	3.80%
		150	29.5	0.00%	0.34%	3.51%
		200	38	-1.30%	-1.81%	3.54%
VII	40	50	7.5	1.35%	1.35%	10.29%
		100	12.7	3.25%	3.25%	10.43%
		150	16.6	5.06%	3.75%	15.28%
		200	24.2	-0.82%	-2.42%	6.61%
VIII	100	50	9.3	1.09%	1.09%	6.90%
		100	19	0.53%	-1.04%	3.26%
		150	24.8	1.22%	0.81%	10.22%
		200	31.1	1.63%	0.97%	10.28%
Total			**736.4**	**-0.23%**	**-0.57%**	**3.89%**

5 CONCLUSIONS

In this paper, we introduced *GASP*, a new framework for multi-dimensional multi-container packing problems. *GASP* combines the simplicity of greedy algorithms with learning mechanisms, aiming to guide the overall method towards good solutions. Extensive computational results both in 2D and 3D bin packing instances showed that *GASP* is able to achieve and sometimes improve state-of-the-art results with a negligible computing effort.

6 ACKNOWLEDGMENTS

While working on this project, the first author was the NSERC Industrial Research Chair on Logistics Management, ESG UQAM, and Adjunct Professor with the Department of Computer Science and Operations Research, Université de Montréal, and the Department of Economics and Business Administration, Molde University College, Norway.

This project was partially funded by the Ministero dell'Istruzione, Università e Ricerca (MIUR) (Italian Ministry of Education, University, and Research), under the Progetto di Ricerca di Interesse Nazionale (PRIN) 2009, "Methods and Algorithms for the Logistics Optimization", and the Natural Sciences and Engineering Council of Canada (NSERC), through its Industrial Research Chair and Discovery Grants programs, and by the partners of the Chair, CN, Rona, Alimentation Couche-Tard and the Ministry of Transportation of Québec.

References

Berkey, J. O. and Wang, P. Y. (1987). Two dimensional finite bin packing algorithms, *Journal of the Operational Research Society* **38**: 423–429.

Bortfeldt, A. (2006). A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces, *European Journal of Operational Research* **172**: 814–?837.

Crainic, T. G., Perboli, G., Rei, W. and Tadei, R. (2011). Efficient lower bounds and heuristics for the variable cost and size bin packing problem, *Computers & Operations Research* **38**: 1474–1482.

Crainic, T. G., Perboli, G. and Tadei, R. (2008). Extreme point-based heuristics for three-dimensional bin packing, *INFORMS Journal on Computing* **20**: 368?–384.

Crainic, T. G., Perboli, G. and Tadei, R. (2009). TS^{2}PACK: A two-stage tabu search heuristic for the three-dimensional bin packing problem, *European Journal of Operational Research* **195**: 744?–760.

Crainic, T. G., Perboli, G. and Tadei, R. (in press). Recent advances in multi-dimensional packing problems, *New Technologies: Trends, Innovations and Research*, InTech. ISBN: 979-953-307-654-5.

Faroe, O., Pisinger, D. and Zachariasen, M. (2003). Guided local search for the three-dimensional bin packing problem, *INFORMS, Journal on Computing* **15**(3): 267–283.

Fekete, S. P. and Schepers, J. (2004). A combinatorial characterization of higher-dimensional orthogonal packing., *Mathematics of Operations Research* **29**(2): 353–368.

Hadjiconstantinou, E. and Iori, M. (2007). A hybrid genetic algorithm for the two-dimensional single large object placement problem, *European Journal of Operational Research* **183**: 1150–1166.

Lodi, A., Martello, S. and Vigo, D. (1999). Approximation algorithms for the oriented two-dimensional bin packing problem, *European Journal of Operational Research* **112**: 158–166.

Lodi, A., Martello, S. and Vigo, D. (2004). Tspack: A unified tabu search code for multi-dimensional bin packing problems, *Annals of Operations Research* **131**: 203–213.

Martello, S., Pisinger, D. and Vigo, D. (2000). The three-dimensional bin packing problem, *Operations Research* **48**(2): 256–267.

Martello, S. and Toth, P. (1990). *Knapsack Problems - Algorithms and computer implementations*, John Wiley & Sons, Chichester, UK.

Martello, S. and Vigo, D. (1998). Exact solution of the finite two dimensional bin packing problem, *Management Science* **44**(44): 388–399.

Standard Performance Evaluation Corporation (2006). SPEC CPU2006 benchmarks. http://www.spec.org/cpu2006/results/.
URL: *http://www.spec.org/cpu2006/results/*

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