An efficient meta-heuristic for multi-dimensional multi-container packing

An Efficient Metaheuristic for Multi-Dimensional Multi-Container Packing

Guido Perboli, Teodor Gabriel Crainic and Roberto Tadei

***Abstract*— In this paper, we introduce *GASP - Greedy Adaptive Search Procedure*, a metaheuristic able to efficiently address two and three-dimensional multiple container packing problems. *GASP* combines the simplicity of greedy algorithms with learning mechanisms aimed to guide the overall method towards good solutions. Extensive experiments indicate that *GASP* attains near-optimal solutions in very short computational times, and improves state-of-the-art results in comparable computational times.**

I. INTRODUCTION

Multi-dimensional, multi-container packing problems play a central role in planning freight transportation and supply chain systems in order to reduce costs and achieve a better use of facilities and equipment. They are encountered as support to operational decisions or as part of more complex tactical decision processes.

Although the importance of the issue is acknowledged, research on multi-dimensional multi-container packing problems is relatively recent [8], state-of-the-art methods requiring high computational times for relatively small instances [3]. Therefore, there is a need for accurate and fast solution methods able to deal with relatively large instances. This is the goal of 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 aimed to guide the overall method towards good solutions. Extensive experiments indicate that *GASP* attains near-optimal solutions in very short computational times, and improves state-of-the-art results in comparable computational times.

II. PROBLEM STATEMENT

We consider the two and three-dimensional multi-container packing problems with a homogeneous set of containers. This setting covers the case where the firm already owns its containers (and all equivalent settings where acquisition or leasing costs are already accounted for) and spans a large amount of practical situations. The cost of the containers is not part of the decision process in such settings, the goal being the minimization of the number of containers. These are the hypotheses of most contributions in the literature and they allow us to compare our results to the state-of-the-art methodologies.

The problem can be formulated as a multi-dimensional bin packing problem and we focus in particular on three and two-dimensional formulations. 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 the items into the minimum number of bins. According to the typology introduced in [12], the problem is also known as the *Three-Dimensional Single Bin-Size Bin Packing Problem* (*3D-SBSBPP*).

In several freight transportation applications, one cannot pile the items. 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 and it is detailed and analyzed in the following sections.

III. LITERATURE REVIEW

$TSPACK$ is the tabu search algorithm for the *2D-BPP* developed by Lodi, Martello, and Vigo [6]. 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 produce guillotine packings, so does the overall algorithm. The algorithm is presently the best metaheuristic for *2D-BPP*, but it requires a computation effort of the order of 60 CPU seconds per instance to achieve these results.

G. Perboli is Assistant Professor, Politecnico di Torino, Torino, Italy, and Associate Member, CIRRELT, Montreal, Canada guido.perboli@polito.it

T. G. Crainic is Professor, UQAM, and Regular Member, CIRRELT, Montreal, Canada TeodorGabriel.Crainic@CIRRELT.ca

R. Tadei is Professor, Politecnico di Torino, Torino, Italy roberto.tadei@polito.it

The same authors [7] presented a shelf-based heuristic for the *2D-BPP*, called *Height first - Area second* (*HA*). 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 by Martello and Toth [9] for the *1D-BPP* problem. The second solution is obtained by ordering the items by non-increasing area of their base and new layers are built. 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 first exact method for the *3D-BPP* was a two-level Branch-and-Bound proposed by Martello, Pisinger, and Vigo [8]. 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. The authors also gave the results of their method by limiting its computational effort to 1000 CPU seconds.

Faroe and Pisinger [4] 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 heuristic, the algorithm iteratively decreases the number of bins, each time using GLS to search for a 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 2 and 3-dimensional instances with up to 200 items. The results were satisfactory, but required of the order of 1000 CPU seconds to be reached.

Crainic, Perboli, and Tadei [2] defined the Extreme Points (EPs) rule of identifying possible positions to place items into a given (partially loaded) bin. The EPs extend the Corner Points of [8] to better exploit the bin volumes, 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) heuristic, producing the *EP-BFD* heuristic for the *1D-BPP*. Extending the *EP-BFD* to address the *3D-BPP* proved far from trivial however, as the ordering of items 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 *C-EPBFD*, a composite heuristic based on *EP-BFD*. Extensive experimental results showed *C-EPBFD* requiring negligible computational efforts and outperforming both current constructive heuristics for the *3D-BPP* and more complex methods, e.g., the truncated Branch-and-Bound by [8].

Crainic, Perboli, and Tadei [3] proposed TS^2PACK, a two-level tabu search metaheuristic 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 made use of the Interval Graph representation of the packing by Fekete and Schepers [5] to reduce the search space. The accuracy of the overall metaheuristic algorithm 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 quite slow convergence rate, requiring of the order of 300 CPU seconds to find the best solution.

This review emphasizes that the state-of-the-art methods available in the literature lack in either accuracy or efficiency. The method we propose aims to address this challenge.

IV. THE PROPOSED SOLUTION METHOD

The main idea of the method we propose is to separate how items are packed, the *feasibility* phase, from the selection of the order according to which the items are packed, the *optimality* phase [3].

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 value of an item relative to the others. We embed these elements into a metaheuristic framework, which provides a learning mechanism used to update the scores.

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

- Build an initial solution by means of *C-EPBFD* and set it as the best solution BS;
- Scoring Phase
  - Initialize the scores (*Score Initialization* procedure in Fig. 1);
  - While *Stopping Conditions* are not encountered, repeat the following steps:
    - Sort the items according to their scores 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* pro-

cedure; otherwise, update the scores using the *Score Update* procedure applied to CS;
  - ∗ If CS is better than BS, then set BS to CS;
  - ∗ The *Parameter Update* procedure then internally adjusts the parameters.
- *Stopping Conditions*. If the value of the best solution is proven to be optimal (i.e., equal to the optimum or to the best lower bound available in the literature), or when a given time limit is reached.

We now detail the different procedures.

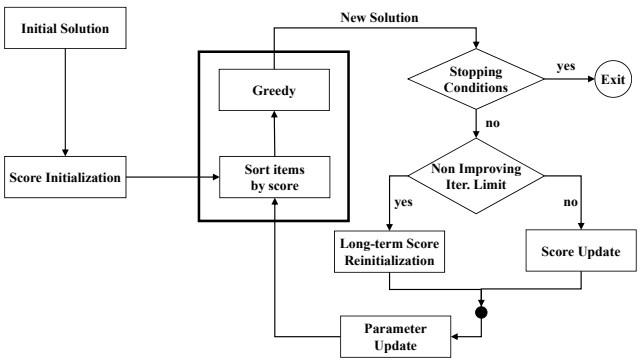

Fig. 1. General scheme of *GASP*

A. Greedy

We select *EP-BFD* as our greedy procedure. The choice is justified by the fact that it is currently the best constructive heuristic available [2].

The procedure builds a feasible multi-container packing by accommodating the items according to the given ordered list. The procedure first tries to load the item into an existing bin. Bins are selected by maximizing a *merit function* measuring both the residual space of the bin (i.e., the free space defined inside a packing by the shape of the items already loaded) and the position where the item can be accommodated, provided the residual space is sufficient to accept it. If the item cannot be loaded into any of the existing bins, a new bin is created. Items are packed according to the *EP*s defined by Crainic, Perboli, and Tadei [2].

B. Initialization

The initial solution is generated by *C-EPBFD*. As recalled in Section III, *C-EPBFD* combines different sorting rules by returning the best solution. Given the item ordering associated to the best 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.

C. 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 ordering of items 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:

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

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 (1) penalizes the items loaded in the well-packed bins and helps, by increasing their score, the items loaded in 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 (1), 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 heuristic.

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 two embedded directions. The first level starts with the largest value of m and gradually decreases it by decreasing the maximum percentage of score variation. For each maximum score-variation value, the second level gradually reduces the number of items that can be swapped in the list. This is implemented by making m depend 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

$$m = \frac{\overline{s}}{p}(k_{max} - k), \tag{2}$$

yielding the following expression for the score updating rule

$$s_i = \begin{cases} s_i \left(1 - \frac{\overline{s}}{p}(k_{max} - k)\right) & b(i) \in \left\{1, .., \lfloor \frac{B}{2} \rfloor\right\}, \\ s_i \left(1 + \frac{\overline{s}}{p}(k_{max} - k\right) & \text{otherwise}, \end{cases} \tag{3}$$

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

D. Long-term Score Reinitialization

Similar to the *Score Initialization* procedure, but starting from the list of the items 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.

E. Parameter Update

The two parameters are dynamically updated:

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

V. COMPUTATIONAL RESULTS

Experiments were carried out on standard benchmark instances. For *2D-BPP*, we consider ten classes of instances from [1] (Classes I-VI) and [10] (Classes VII-X), the rationale in building them being similar to that for the *3D-BPP* instances. For each class, we consider 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, Pisinger, and Vigo [8] proposed seven classes of instances for the *3D-BPP*. For Classes *I* to *V*, the bin size is $W = H = D = 100$ and the items belong to five types of items, ranging from small to large-sized items. The five classes mix the item types in order to test different usage scenarios. In classes *VI* to *VIII* bin and item dimensions vary according to the following rules:

- *Class VI*: w_i, l_i, $h_i \sim$ U[1,10] and $W = L = H = 10$;
- *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 [8], [4], and [3], we do not consider Classes II and III, which display properties similar to those of Class I. For each remaining class, i.e., I and IV to VIII, we consider instances with 50, 100, 150, and 200 items. Given a class and an instance size, we generate 10 different problem instances based on different random seeds.

GASP has been 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 time limit was set to 3 and 5 seconds for 2D and 3D problems, respectively, while the values of the other parameters are $\overline{s} = 0.1$, $k_{max} = 4$, $k = 1$, and $p = 1$. The *Long-term Score Reinitialization* procedure is applied every 1000 iterations.

A. 2D-BPP results

We compared the results of *GASP* to those of $TSPACK$, the tabu search of Lodi, Martello, and Vigo [6], as well as to the best results from the literature obtained by heuristics 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 [6]. A 3-second time limit was given to *GASP*.

The results are summarized in Table I. 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 been generally obtained by means of different exact methods and with a computational 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 computational effort by more than one order of magnitude. Moreover, *GASP* reaches results that are less than 1% from the overall optima. We want to stress that these optima are obtained by different methods, and most

of them come from exact methods requiring a significant computational effort.

TABLE I

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**

TABLE II

3D-BPP: COMPARISON OF *GASP* AND STATE-OF-THE-ART METHODS

Class	Bins	n	GASP 5 ss	MPV 1000 sec	GLS 1000 sec	TS^{2}PACK 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%**

B. 3D-BPP results

For the 3D case, *GASP* is compared to GLS [4], MPV, the truncated *Branch and Bound* proposed in [8], and TS^2PACK [3].

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 benchmarks [11]). A time limit of 5 CPU seconds per 3D problem was allocated to *GASP*, to better represent circumstances when 300 second computational times are not acceptable.

TABLE III

3D-BPP: COMPARISON OF *GASP* AND STATE-OF-THE-ART METHODS IN COMPARABLE COMPUTATIONAL TIMES

Class	Bins	n	GASP 5 ss	GLS 60 ss	TS^{2}PACK 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%**

Table II 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 the *GASP* heuristic and the method compared to, 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 & Bound and has a gap of only 0.9% with the best algorithm in the literature, with a negligible computational time: 5 CPU seconds compared to 1000 for GLS and 300 for TS^2PACK.

To further illustrate this efficiency, Table III displays the performance of *GASP* w.r.t. those of GLS and TS^2PACK, in comparable computational 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 [11]). These results are impressive as *GASP* actually improves the solutions of both GLS and TS^2PACK up to 0.6% on average.

VI. 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 aimed to guide the overall method towards good solutions. Extensive computational results both on 2D and 3D bin packing instances showed that *GASP* is able to achieve and sometimes improve state-of-the-art results with a negligible computational effort.

VII. ACKNOWLEDGMENTS

Funding for this project has been provided by the Italian Ministry of University and Research, under the 2007 PRIN Project "Optimization of Distribution Logistics" and the Natural Sciences and Engineering Council of Canada (NSERC), through its Industrial Research Chairs and Discovery Grants programs.

REFERENCES

[1] J. O. Berkey and P. Y. Wang, "Two dimensional finite bin packing algorithms," *Journal of the Operational Research Society*, vol. 38, pp. 423–429, 1987.

[2] T. Crainic, G. Perboli, and R. Tadei, "Extreme point-based heuristics for three-dimensional bin packing," *INFORMS Journal on Computing*, vol. 20, pp. 368–384, 2008.

[3] ——, "TS^{2}PACK: A two-stage tabu search heuristic for the three-dimensional bin packing problem," *European Journal of Operational Research*, vol. 195, pp. 744–760, 2009.

[4] O. Faroe, D. Pisinger, and M. Zachariasen, "Guided local search for the three-dimensional bin packing problem," *INFORMS, Journal on Computing*, vol. 15, no. 3, pp. 267–283, 2003.

[5] S. P. Fekete and J. Schepers, "A combinatorial characterization of higher-dimensional orthogonal packing." *Mathematics of Operations Research*, vol. 29, no. 2, pp. 353–368, 2004.

[6] A. Lodi, S. Martello, and D. Vigo, "Approximation algorithms for the oriented two-dimensional bin packing problem," *European Journal of Operational Research*, vol. 112, pp. 158–166, 1999.

[7] ——, "Tspack: A unified tabu search code for multi-dimensional bin packing problems," *Annals of Operations Research*, vol. 131, pp. 203–213, 2004.

[8] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, vol. 48, no. 2, pp. 256–267, 2000.

[9] S. Martello and P. Toth, *Knapsack Problems - Algorithms and computer implementations*. Chichester, UK: John Wiley & Sons, 1990.

[10] S. Martello and D. Vigo, "Exact solution of the finite two dimensional bin packing problem," *Management Science*, vol. 44, no. 44, pp. 388–399, 1998.

[11] Standard Performance Evaluation Corporation, "SPEC CPU2006 benchmarks," 2006, http://www.spec.org/cpu2006/results/. [Online]. Available: http://www.spec.org/cpu2006/results/

[12] G. Wäscher, H. Haussner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, pp. 1109–1130, 2007.