BPPLIB: A Library for Bin Packing and Cutting Stock Problems

Maxence Delorme[(1)], Manuel Iori[(2)], Silvano Martello[(1)]

[(1)] DEI "Guglielmo Marconi", University of Bologna
[(2)] DISMI, University of Modena and Reggio Emilia
maxence.delorme2@unibo.it, manuel.iori@unimore.it, silvano.martello@unibo.it

Abstract

The bin packing problem (and its variant, the cutting stock problem) is among the most intensively studied combinatorial optimization problems. We present a library of computer codes, benchmark instances, and pointers to relevant surveys for these two problems. The computer code section includes twelve programs: seven are directly downloadable from the library page, while for the remaining five we provide addresses where they can be obtained or downloaded. Some of the codes for which we provide an original C++ implementation need an ILP solver. For such cases, the library provides two versions: one that uses the commercial solver Cplex, and one that uses the freeware solver SCIP. The benchmark section provides over six thousands instances (partly coming from the literature and partly randomly generated), together with the corresponding solutions. Instances that are difficult to solve to proven optimality are included. The library also includes a BibTeX file of more than 150 references on this topic. We conclude this work by reporting the results of new computational experiments on a number of computer codes and benchmark instances.

Keywords: Bin packing, Cutting stock, Computer codes, Benchmark instances, Surveys.
MSC Codes: 90-00, 90-08, 90-C10, 90-C27.

1 Introduction

In the *bin packing problem* (BPP), n *items* of given integer *weight* w_j $(j = 1, \ldots, n)$ have to be packed into the minimum number of identical containers (*bins*) of integer *capacity* c. Let u be any upper bound on the solution value. Let us introduce two sets of binary variables: y_i $(i = 1, \ldots, u)$, taking the value one if and only if bin i is used in the solution, and x_{ij} $(i = 1, \ldots, u; j = 1, \ldots, n)$, taking the value one if and only if item j is packed into bin i. A possible simple *Integer Linear Programming* (ILP) model of the problem is then (see Martello and Toth [25])

$$\min \quad \sum_{i=1}^{u} y_i \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \leq c y_i \quad (i = 1, \ldots, u), \tag{2}$$

$$\sum_{i=1}^{u} x_{ij} = 1 \qquad (j = 1, \ldots, n), \tag{3}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, u), \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad (i = 1, \ldots, u; j = 1, \ldots, n). \tag{5}$$

Among the many variants and generalizations of the problem, the most intensively studied is probably the *Cutting Stock Problem* (CSP). In this case, instead of single items, we have m *item types* of weight w_j and an integer *demand* d_j $(j = 1, \ldots, m)$ per item type. The objective is to pack d_j copies of each item type j into the minimum number of bins. By introducing an additional set of integer variables ξ_{ij} $(i = 1, \ldots, u; j = 1, \ldots, m)$ giving the number of items of type j packed into bin i, the CSP can be modeled by the ILP

$$\min \quad \sum_{i=1}^{u} y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \leq c y_i \quad (i = 1, \ldots, u), \tag{7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \qquad (j = 1, \ldots, m), \tag{8}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, u), \tag{9}$$

$$\xi_{ij} \geq 0, \text{integer} \quad (i = 1, \ldots, u; j = 1, \ldots, m). \tag{10}$$

The BPP is known to be $\mathcal{NP}$-hard in the strong sense (by transformation from the 3-Partition problem, see Garey and Johnson [18]). As any instance of either problem can easily be transformed into an equivalent instance of the other, the same holds for the CSP.

These two problems are among the most intensively studied problems in combinatorial optimization. Two recent surveys on exact methods (Delorme, Iori, and Martello [12]) and approximation algorithms (Coffman, Csirik, Galambos, Martello, and Vigo [9]) consider in total over 230 different references. Previous surveys were presented by Garey and Johnson [19], Coffman, Garey, and Johnson [10, 11], Sweeney and Paternoster [31], Dyckhoff [14], Martello and Toth [25] (Chapter 8), Dyckhoff and Finke [15], Valério de Carvalho [34], Wäscher, Haußner, and Schumann [36], among others. Most solution methodologies have been tried on these problems: different kinds of ILP models, lower bound computations, branch-and-bound, branch-and-price, constraint programming, approximation algorithms, heuristics, and metaheuristics.

In this paper we describe the BPPLIB, a library for bin packing and cutting stock problems. An earlier version of the current library was implemented as an auxiliary instrument for the computational experiments presented in [12]. As most other web-based libraries for optimization problems (see, e.g., Beasley [2], Burkard, Karisch, and Rendl [5, 6], Applegate, Bixby, Chvatal, and Cook [1] - the well-known *TSP web page* `http://www.math.uwaterloo.ca/tsp/`, Groër, Golden, and Wasil [20], Koch et al. [22], Friberg [17]), and Uchoa et. al [32]), the current BPPLIB page contains pointers to the literature, pointers to computer codes, and benchmark instances. In addition, it provides a number of freeware codes that were specifically implemented by the authors.

In the next section we introduce the computer codes provided by the BPPLIB. In Section 3 we describe the available benchmarks: some of them were used in [12] for the computational evaluations of the different exact approaches, using Cplex when needed. As the library has been enriched by also providing versions based on SCIP, in Section 4 we provide new experiments aiming at evaluating the computational difference between the two versions. In addition, we describe new test instances, that appeared after the publication of [12], and present the corresponding computational experiments.

2 Computer codes

The BPPLIB provides twelve computer codes of different types for the exact solution of the BPP and the CSP.

Branch-and-bound

The first effective exact algorithms for the BPP were based on a branch-and-bound approach. The library provides, in chronological order:

- **MTP**: Fortran code of the BPP algorithm by Martello and Toth [25], originally available in the diskette accompanying the book. The algorithm adopts a depth-first strategy to explore a branch-decision tree that considers one item per level: descendant nodes are generated by assigning the current item, in turn, to all already initialized bins and possibly to a new bin. While the approach is effective for BPP instances, considering one item at a time is clearly inefficient for CSP instances with high item multiplicity. The code can be run using the Fortran front end of the GNU Compiler Collection GCC;

- **BISON**: Scholl, Klein, and Jürgens [29] obtained a very efficient BPP algorithm by enriching MTP through new lower bounds and a Tabu search algorithm to help the search by means of effective heuristic solutions. The code was implemented in Pascal, and can be obtained from the authors, using the address provided in the library. Worth is mentioning that, in spite of its 'age', this program is still working and quite effective (see [12]): at the time of writing, it can be run using compiler `fpc (version 3.0.0 for x86_64)`;

- **CVRPSEP**: we provide a link to the C code implemented by J. Lysgaard as part of a separation routine within the algorithm by Lysgaard, Letchford, and Eglese [24] for the capacitated vehicle routing problem. The routine was obtained by using procedures from MTP. It is generally less efficient than MTP, but we decided to include it in the library mainly because one may prefer a C code to a Fortran code. The implementation details can be found in a technical report by Lysgaard [23].

Branch-and-price

This modern evolution of the branch-and-bound approach can produce very effective algorithms for the problems at hand. We provide links to two computer codes:

- **BELOV**: C++ implementation by G. Belov of the algorithm by Belov and Scheithauer [3], using Cplex for the inner routines. The algorithm is tailored to the exact solution of CSP instances, and it computationally proved to be the most powerful approach both in the case of low and high item multiplicity;

- **SCIP-BP**: freeware SCIP C code for a branch-and-price BPP algorithm based on the classical Ryan and Foster [27] branching rule and available at the SCIP web page. This code is only effective for instances with small number of item types and low item multiplicity.

Pseudo-polynomial formulations solved via ILP

Already in the Seventies, pseudo-polynomial models coming from a graph representation of the solution space were proposed. For many years, solution approaches based on such models have been regarded as very theoretical, with no practical interest, due to the huge number of variables and constraints they imply. Up to few years ago, these methods were mainly used within branch-and-price algorithms (see, e.g., Valério de Carvalho [33]). However, nowadays computational power of ILP solvers made them competitive with branch-and-price algorithm also for the case of realistic size instances, provided the number of generated variables (that depends on capacity, number of items, and item weights) is not too big. The BPPLIB provides four algorithms based on pseudo-polynomial models:

- **ONECUT**: C++ implementation of the *one-cut* CSP model independently defined in the Seventies by Rao [26], and Dyckhoff [13];
- **ARCFLOW**: C++ implementation of the *arc-flow* CSP model by Valério de Carvalho [33];
- **DPFLOW**: C++ implementation of the *DP-flow* BPP model by Cambazard and O'Sullivan [7];
- **VPSOLVER**: link to the C++ implementation by Brandão and Pedroso [4] of their CSP algorithm. This is currently the most effective pseudo-polynomial approach, and its performance is often competitive with that of BELOV.

For the first three codes we provide both a version that uses Cplex as an inner routine, and a version that uses SCIP. Code VPSOLVER was instead implemented by the authors in a version that invokes Gurobi.

3 Benchmarks

The BPPLIB provides in total 6 195 test instances belonging to four categories. Each instance is provided, using unified formats, both in BPP and CSP version.

Literature instances

This section contains the 1 615 instances proposed by

- Falkenauer [16]: 80 (easy) instances with uniformly distributed item sizes and 80 (more difficult) instances obtained through triplets of items that must be packed into the same bin in any optimal solution;
- Scholl, Klein, and Jürgens [29]: three sets of instances with uniformly distributed item sizes. The first set is composed by 720 easy instances, the second set by 480 instances of medium difficulty, and the third set by 10 difficult instances characterized by huge capacities;
- Wäscher and Gau [35]: 17 very hard instances selected by the authors from a much larger set of instances belonging to different typologies;

- Schwerin and Wäscher [30]: two sets of 100 relatively easy instances each;
- Schoenfield [28]: 28 hard instances that do not involve huge capacities.

In the library, each set is identified by the name of the (first) author.

Randomly generated instances

The library provides the 3 840 instances that were randomly generated for the computational experiments reported in [12]. The instances have different values of n (50, 100, 200, 300, 400, 500, 750, 1 000), of c (50, 75, 100, 120, 125, 150, 200, 300, 400, 500, 750, 1 000), and of the minimum ($0.1\,c$, $0.2\,c$) and maximum ($0.7\,c$, $0.8\,c$) item weight. The benchmark contains 10 instances for each of the 384 quadruplets (n, c, minimum weight, maximum weight). These instances are relatively easy, and the algorithms listed in Section 2 could solve most of them within reasonable CPU times.

Hard instances

In order to perform experiments on challenging instances, a number of so called *augmented Non-IRUP* and *augmented IRUP* instances were proposed in [12], using as a basis a set of Non-IRUP instances presented in Caprara, Dell'Amico, Díaz Díaz, Iori, and Rizzi [8]. For the 250 instances of the former class an optimal solution is easy to find, but its optimality is very difficult to prove. Even the continuous relaxation of the set covering formulation (the basis of branch-and-price algorithms) and that of the pseudo-polynomial formulations fail in reaching the optimal value. As a consequence, algorithms based on such relaxations require either a huge branching process or a heavy cut generation: already for $n \approx 400$, no algorithm is capable of solving all of them to proven optimality. For the 250 instances of the latter class, it is easy to produce a lower bound whose value is equal to the optimum, but it is difficult to build an optimal solution.

GI instances

The library includes 240 new instances, proposed by Gschwind and Irnich [21] after the publication of [12]. Such instances, uniformly randomly generated, are characterized by very large capacities. They are organized into four sets of 60 instances each. As shown in the next section, two of such sets are generally difficult to solve.

4 Computational experiments

We report the results of some experiments executed on an Intel Xeon 3.10 GHz (equipped with four cores) with 8 GB RAM, all executed with a single core. In order to test the codes on non-trivial instances, we preliminarily obtained an upper bound through the classical *best fit decreasing heuristic* and computed the lower bound value known as L2 (see [24]): only instances for which these two values were different were then tested.

Tables 1 and 2 give the number of literature instances that were solved in one CPU minute (and, in parentheses, the average CPU time), by, respectively, the enumeration algorithms and the pseudo-polynomial models. (For the non-solved instances, one CPU minute was

Table 1: Literature instances, enumerative algorithms. Number of instances solved in less than one minute (average CPU time in seconds).

Set	Number of tested instances	Branch-and-bound			Branch-and-price	
		MTP	BISON	CVRPSEP	BELOV	SCIP-BP
Falkenauer U	74	22 (42.8)	44 (24.5)	22 (42.2)	**74** (0.0)	18 (50.1)
Falkenauer T	80	6 (55.5)	42 (30.6)	0 (60.0)	57 (24.7)	35 (39.4)
Scholl1	323	242 (15.1)	288 (7.0)	223 (19.4)	**323** (0.0)	244 (22.4)
Scholl2	244	130 (28.2)	233 (3.0)	65 (44.2)	**244** (0.3)	67 (49.2)
Scholl3	10	0 (60.0)	3 (42.0)	0 (60.0)	**10** (14.1)	0 (60.0)
Wäscher	17	0 (60.0)	10 (24.7)	0 (60.0)	**17** (0.1)	0 (60.0)
Schwerin1	100	15 (51.1)	100 (0.0)	9 (55.4)	**100** (1.0)	0 (60.0)
Schwerin2	100	4 (57.6)	63 (22.2)	0 (60.0)	**100** (1.4)	0 (60.0)
Hard28	28	0 (60.0)	0 (60.0)	0 (60.0)	**28** (7.3)	7 (51.2)
Total (average)	976	419 (34.4)	783 (12.3)	319 (40.8)	953 (2.7)	371 (42.2)

Table 2: Literature instances, pseudo polynomial models. Number of instances solved in less than one minute (average CPU time in seconds).

Set	Number of tested instances	ONECUT		ARCFLOW		DPFLOW		VPSOLVER
		Cplex	SCIP	Cplex	SCIP	Cplex	SCIP	
Falkenauer U	74	**74** (0.2)	67 (23.8)	**74** (0.2)	70 (18.7)	37 (38.8)	0 (60.0)	**74** (0.1)
Falkenauer T	80	**80** (8.7)	21 (44.9)	**80** (3.5)	33 (41.4)	40 (41.7)	20 (50.8)	**80** (0.4)
Scholl1	323	**323** (0.1)	318 (5.0)	**323** (0.1)	320 (5.1)	289 (13.0)	178 (34.0)	**323** (0.1)
Scholl2	244	118 (38.7)	20 (56.3)	202 (18.9)	39 (53.7)	58 (50.4)	11 (58.5)	208 (14.0)
Scholl3	10	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	**10** (6.3)
Wäscher	17	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	0 (60.0)	6 (49.4)
Schwerin1	100	**100** (13.1)	0 (60.0)	**100** (1.5)	0 (60.0)	0 (60.0)	0 (60.0)	**100** (0.3)
Schwerin2	100	**100** (11.7)	0 (60.0)	**100** (1.5)	1 (59.5)	0 (60.0)	0 (60.0)	**100** (0.3)
Hard28	28	6 (54.6)	0 (60.0)	16 (40.6)	0 (60.0)	0 (60.0)	0 (60.0)	27 (14.2)
Total (average)	976	801 (16.3)	426 (36.9)	895 (8.2)	463 (35.6)	424 (38.9)	209 (50.3)	928 (5.0)

considered.) For each instance set, boldface highlights the cases where all instances were solved to proven optimality.

The results in Table 1 summarize the (much more detailed) tables presented in [12]: they are provided here in order to give the reader information on the performance of the codes provided in the BPPLIB. The results in Table 2 include new results obtained using SCIP as the ILP solver. The tables confirm the clear superiority of BELOV and VPSOLVER over the other algorithms.

Table 2 shows in addition that the performance of ONECUT and ARCFLOW is not affected by the ILP solver for most of the easy instances, while their performance for the difficult instances sharply worsens when SCIP is used instead of Cplex. This behavior could be explained by the difference in the number of generated variables and constraints between different benchmarks. For example, ARCFLOW produces, on average, 1 735 variables and 103 constraints for Scholl 1 instances, while for "Scholl 2" it produces, on average, 39 307 variables and 840 constraints.

Tables 3 and 4 refer to the randomly generated instances used in [12], and provide the same information as in Tables 1 and 2. The previous observations are confirmed: BELOV and VPSOLVER outperform the other approaches, and the use of SCIP decreases the algorithms'

performance, especially for large values of n.

Table 3: Random instances, enumerative algorithms. Number of instances solved in less than one minute (average CPU time in seconds).

n	Number of tested instances	Branch-and-bound			Branch-and-price	
		MTP	BISON	CVRPSEP	BELOV	SCIP-BP
50	165	163 (0.8)	**165** (0.0)	164 (0.4)	**165** (0.0)	**165** (0.9)
100	271	243 (7.4)	257 (3.8)	239 (8.4)	**271** (0.0)	**271** (4.6)
200	359	237 (21.6)	290 (12.0)	220 (25.0)	**359** (0.0)	293 (22.6)
300	393	166 (35.7)	265 (20.7)	144 (38.7)	**393** (0.1)	155 (44.1)
400	425	151 (39.1)	244 (26.1)	138 (41.2)	**425** (0.2)	114 (49.8)
500	414	121 (43.0)	208 (30.3)	128 (42.6)	**414** (0.2)	69 (55.1)
750	433	93 (47.3)	214 (30.9)	98 (47.3)	**433** (0.4)	22 (59.5)
1000	441	78 (49.5)	196 (33.9)	73 (50.8)	**441** (0.7)	0 (60.0)
Total (average)	2901	1252 (34.7)	1839 (22.6)	1204 (36.0)	**2901** (0.2)	1089 (42.4)

Table 4: Random instances, pseudo polynomial models. Number of instances solved in less than one minute (average CPU time in seconds).

n	Number of tested instances	ONECUT		ARCFLOW		DPFLOW		VPSOLVER
		Cplex	SCIP	Cplex	SCIP	Cplex	SCIP	
50	165	**165** (0.1)	163 (2.0)	**165** (0.1)	**165** (1.6)	**165** (0.5)	162 (5.1)	**165** (0.0)
100	271	**271** (0.8)	249 (8.6)	**271** (0.3)	262 (10.1)	**271** (5.0)	168 (34.5)	**271** (0.1)
200	359	358 (2.4)	286 (15.4)	**359** (0.8)	278 (20.2)	292 (21.0)	76 (51.8)	**359** (0.3)
300	393	385 (4.5)	272 (22.2)	391 (2.0)	262 (24.9)	243 (33.9)	31 (57.3)	**393** (0.6)
400	425	408 (5.1)	293 (22.0)	421 (3.0)	276 (25.8)	193 (42.4)	23 (58.1)	**425** (0.8)
500	414	394 (6.3)	275 (24.0)	402 (4.0)	258 (26.5)	169 (44.8)	13 (58.8)	413 (1.7)
750	433	401 (7.8)	284 (24.3)	415 (6.0)	279 (25.7)	120 (52.6)	12 (59.1)	431 (2.4)
1000	441	407 (8.1)	280 (25.8)	416 (6.8)	281 (26.1)	67 (56.4)	7 (59.6)	434 (3.4)
Total (average)	2901	2789 (5.0)	2102 (20.0)	2840 (3.3)	2061 (22.3)	1520 (36.7)	492 (52.5)	2891 (1.4)

4.1 GI instances

We report in Table 5 the results of computational experiments for the GI benchmark, a set of CSP instances recently proposed by Gschwind and Irnich [21] for testing their dual inequalities aimed at stabilizing column generation processes. They are organized into four groups (AA, AB, BA, and BB), characterized by different item weight ranges and capacities. Each group has three sets of 20 instances each, characterized by the number of item types (125, 250, and 500). We tested the best enumerative algorithm (BELOV) and the best pseudo-polynomial approaches (ARCFLOW and VPSOLVER) with a time limit of one hour. BELOV could solve all of these instances very quickly, while they turned out to be extremely difficult for the pseudo-polynomial models. The behavior of the latter approaches was particularly poor for the instances that have items with very small weight and huge capacities (AB and BB, with $c \geq 500\,000$), which induce a high number of variables and constraints. For example, the ILP model produced by ARCFLOW has on average 549 441 variables and 131 219 constraints for instances AA with $m = 125$, but 5 754 617 variables and 404 283 constraints for instances AB with $m = 125$.

Table 5: Number of GI instances solved in less than one hour (average time in seconds).

Set	m	Number of tested instances	BELOV	ARCFLOW	VPSOLVER
AA	125	20	**20** (0.1)	19 (1 092.6)	**20** (0.9)
	250	20	**20** (0.9)	0 (3 600.0)	**20** (14.5)
	500	20	**20** (7.5)	0 (3 600.0)	16 (1 345.9)
AB	125	20	**20** (0.7)	0 (3 600.0)	0 (3 600.0)
	250	20	**20** (2.1)	0 (3 600.0)	0 (3 600.0)
	500	20	**20** (29.9)	0 (3 600.0)	0 (3 600.0)
BA	125	20	**20** (0.1)	**20** (1 120.9)	**20** (1.4)
	250	20	**20** (1.3)	0 (3 600.0)	**20** (23.1)
	500	20	**20** (7.2)	0 (3 600.0)	17 (1 450.2)
BB	125	20	**20** (0.2)	0 (3 600.0)	0 (3 600.0)
	250	20	**20** (2.3)	0 (3 600.0)	0 (3 600.0)
	500	20	**20** (29.1)	0 (3 600.0)	0 (3 600.0)
Total (average)		240	**240** (6.8)	39 (3 234.8)	113 (2 036.3)

References

[1] D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook. *The Traveling Salesman Problem - A Computational Study*. Princeton University Press, Princeton, NJ, 2006.

[2] J. E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[3] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171:85–106, 2006.

[4] F. Brandão and J.P. Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2016.

[5] R.E. Burkard, S.E Karisch, and F. Rendl. QAPLIB– a quadratic assignment problem library. *European Journal of Operational Research*, 55:115 – 119, 1991.

[6] R.E. Burkard, S.E. Karisch, and F. Rendl. QAPLIB– a quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997.

[7] H. Cambazard and B. O'Sullivan. Propagating the bin packing constraint using linear programming. In *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 of *Lecture Notes in Computer Science*, pages 129–136. Springer Berlin Heidelberg, 2010.

[8] A. Caprara, M. Dell'Amico, J.C. Díaz Díaz, M. Iori, and R. Rizzi. Friendly bin packing instances without integer round-up property. *Mathematical Programming*, 150:5–17, 2014.

[9] E.G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Survey and classification. In P.M. Pardalos, D.-Z. Du, and R.L. Graham, editors, *Handbook of Combinatorial Optimization*. Springer New York, 2013.

[10] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing - an updated survey. In G. Ausiello, M. Lucentini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49–106. Springer Vienna, 1984.

[11] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. In D.S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[12] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255:1–20, 2016.

[13] H. Dyckhoff. A new linear programming approach to the cutting stock problem. *Operations Research*, 29:1092–1104, 1981.

[14] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[15] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution*. Physica-Verlag, Heidelberg, 1992.

[16] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *J. of Heuristics*, 2:5–30, 1996.

[17] H.A. Friberg. CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization. *Mathematical Programming Computation*, 8:191–214, 2016.

[18] M.G. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. Freeman, New York, 1979.

[19] M.R. Garey and D.S. Johnson. Approximation algorithms for bin-packing problems: A survey. In G. Ausiello and Lucertini, editors, *Analysis and Design of Algorithms in Combinatorial Optimization*, pages 147–172. Springer Vienna, 1981.

[20] C. Groër, B. Golden, and E. Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2:79–101, 2010.

[21] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, 28:175–194, 2016.

[22] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3:103, 2011.

[23] J. Lysgaard. CVRPSEP: A package of separation routines for the capacitated vehicle routing problem. Technical report, Aarhus School of Business, Denmark, 2003.

[24] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 2004.

[25] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990, (available on line at `www.or.deis.unibo.it`).

[26] M.R. Rao. On the cutting stock problem. *Journal of the Computer Society of India*, 7:35–39, 1976.

[27] D.M. Ryan and B.A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280. North-Holland, 1981.

[28] J.E. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. Technical report, US Army Space and Missile Defense Command, Huntsville, Alabama, USA, 2002.

[29] A. Scholl, R. Klein, and C. Jäurgens. Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24:627–645, 1997.

[30] P. Schwerin and G. Wäscher. The bin-packing problem: a problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4:377–389, 1997.

[31] P.E. Sweeney and E.R. Paternoster. Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43:691–706, 1992.

[32] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257:845–858, 2017.

[33] J.M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999.

[34] J.M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, 2002.

[35] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: a computational study. *Operations-Research-Spektrum*, 18:131–144, 1996.

[36] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130, 2007.