A *Renault* truck loading problem: from benchmarking to improvements

Jean Respen[*] Nicolas Zufferey

Abstract

To deliver goods to car factories, the car manufacturer *Renault* is daily facing a complex truck loading problem where various goods must be packed into a truck such that they fulfill different constraints. As trucks can deliver goods to different factories on the same tour, classes of items have been defined, where a class is associated with a delivery point. As the number of items and the standard deviation of the sizes of the items are significant, no exact algorithm is competitive. We propose four local search methods, namely four tabu search algorithms. Results show that tabu search is competitive compared to the greedy heuristics developed by *Renault*.

Keywords: Truck loading, metaheuristics, bin-packing

1 Introduction

The French car manufacturer *Renault* is facing a complex truck loading problem, where items need to be placed in a truck such that they fulfill different constraints. This problem is called here the *Renault* truck loading problem ($RTLP$). *Renault* is dealing on a daily basis with more than a thousand trucks, which have to deliver goods to car factories, making this problem relevant to their production plan. As a single truck can deliver goods to different delivery points, classes of items have been defined, where a class is associated with a delivery point. Each problem instance contains the size of the truck (in millimeters) and the various sizes of all the items that must fit in (in millimeters). The heights of the items can be ignored as they rely on complex factory constraints which are supposed to be already satisfied. At first sight, this problem seems related to a strip-packing 2D problem with rotation, which has been already covered by many research papers. In the strip-packing 2D problem, rectangular items must be packed in a single bin of fixed width and infinite length, with the objective of minimizing the total length of the packing. Relevant surveys on this topic can be found in [4], [7], [13], and [16]. This problem is a strongly NP-hard combinatorial problem.

In this paper, we aim to tackle some new features proposed by *Renault*: different *classes* of items, and a significant *number* of items per truck in conjunction with a large *standard deviation* of the sizes of the items. Such elements make the considered problem more relevant to modern and realistic issues. To solve $RTLP$, *Renault* proposes a simple but efficient greedy heuristic (SG), and an advanced greedy heuristic called "look-ahead greedy" (LAG). *Renault* proposes us to benchmark their algorithms with state-of-the-art metaheuristics, to measure the possible gap between their algorithms and a possible better solution method. An important aspect is the tradeoff between the computing time and the solution quality. As LAG is fast, another metaheuristic is only relevant if it is fast and finds an important gap in the solution quality.

The paper is organized as follows. In Section 2 is proposed a formal description of the problem. In Section 3, a review of the literature concerning the bin-packing problems is depicted. In Section 4 are proposed the heuristics for $RTLP$ and the obtained results are discussed in Section 5. Section 6 concludes the research and proposes some future works.

2 Description of the problem

$RTLP$ can be formally described as follows: a number n of items, each one belonging to a certain *class* C_i (with $i \in \{1, \ldots, m\}$ such that $m \leq n$), need to be placed in a truck such that all the items belonging to the same class are adjacent. In addition, the classes must be placed in an increasing fashion from the front to the rear of the truck. More formally, the ordinate of the origin item which belongs to class C_i (label 1 on Figure 1) must be strictly smaller than the ordinate of the extremity of any item of class C_{i+1} (label 2 on Figure 1), and such that the ordinate of the extremity item (the closest one to the

[*]Both authors from: HEC - University of Geneva, Switzerland (email: `jean.respen@unige.ch`, `nicolas.zufferey-hec@unige.ch`)

rear) of class C_m (label 3 on Figure 1), denoted as f, is minimized. The truck size is a hard constraint to fulfill, as it is not allowed to exceed neither its length nor its width.

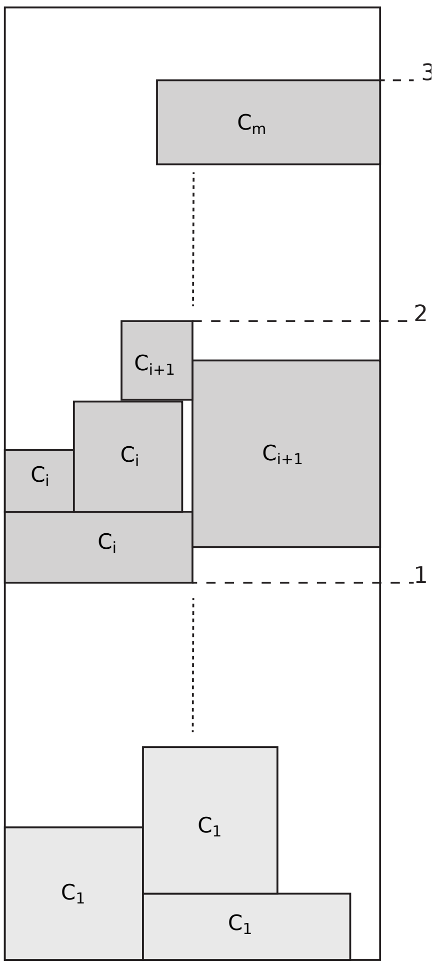

Figure 1: A possible solution

Nowadays, small and medium instances of bin-packing problems are tackled with exact methods and solvers based on linear programming. For the standard bin-packing problem, small instances contain up to 100 items. In contrast, small instances are limited to 10 items for $RTLP$. This is due to the large standard deviation among the sizes of the items (the items have widths or lengths of minimum 570mm and maximum 1810mm with a standard deviation of 196), and of the classes management. *Renault* showed in [12] that no exact method can be competitive to tackle their real instances. Therefore, heuristics and metaheuristics are considered in this paper for $RTLP$.

3 Related literature

For exact methods, literature often proposes methods to tackle instances proposed by Beasley (1985, `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`) or by Bengston (1982, `http://www.ibr.cs.tu-bs.de/alg/packlib/xml/b-prpha-82-xml.shtml`). The sizes of the instances are with up to 200 items. Each item has an width or a length in $\{1, \dots, 100\}$ and a standard deviation of approximately 30 to 40.

If the number of items is less than a hundred, an exact branch-an-bound is proposed in [11] to tackle a problem where a set of n rectangular pieces must be cut from an unlimited number of standardized stock pieces. This problem is a generalization of the 2D *bin-packing* problem (2D-BPP). More recently, in [10], an exact method for the *strip-packing* problem is proposed, solving instances with up to 200 items. The paper proposes a new relaxation, driving to a better lower bound, which is then used with a branch-and-bound algorithm to solve instances to optimality. Lower-bounds and an integer linear formulation for the 2D-BPP with less than a hundred items are depicted in [14], where bins have variable sizes and costs, and the objective is to minimize the overall cost of bins used for packing the items. For the *perfect* packing problem, where there is no wasted space, in [6] is proposed an exhaustive approach, using a branch-and-bound, able to tackle instance with up to 30 items. Finally a mixed integer programming formulation for the *three-stages* 2D-BPP is proposed in [15]. In real application, three-stages often occur, where a stage is either a horizontal or a vertical cut. Because of the specificity of $RTLP$, it is not possible to take advantage of the above exact algorithms to tackle it. *Renault* even showed in [12] that an exact method relying on a recent version of CPLEX is limited to 6 or 7 items.

In the area of bin-packing, many researches focus on heuristics to tackle problems involving packing of items. A survey is proposed in [9] on heuristic and metaheuristic methods for the 2D-BPP, with greedy algorithms and a tabu search for four different problems (bin-packing with or without *rotation*, and with *guillotine* cutting required or not). Instances sizes varies from 10 up to 100. The goal of the tabu search is to empty a specific target bin B by performing moves, which consist in moving subsets of items from B to other bins, where they can be rearranged. A tree-decomposition heuristic for the BPC-2D (2D bin-packing with *conflict*) is proposed in [5] for instances with up to 100 items, where a *conflict graph* $G = (V, E)$ is given, for which each vertex $j \in V$ represents an item, and there is an edge $[j, j'] \in E$ if items j and j' are incompatible. If two incompatible items are loaded in the same bin, a conflict occurs. The goal is to minimize the total number of bins, without creating any conflict. Simple heuristics and a tabu search are proposed to tackle this problem, where for the tabu search, the neighborhood structure consists to move an item from a bin to another. A tabu search is proposed in [8] to tackle the problem of packing each item into a bin, such that the total number of required bins is minimized. A move also corresponds to relocate an item in the solution. Instances contain at most 100 items. The TS^2PACK algorithm is proposed in [1]. It consists in a two-level tabu search for the 3D bin-packing problem (where the height of the bin has to be also considered). Whereas the first level aims at minimizing the total required number of bins, the second level consists in optimizing the packing of the bins. Instances sizes range from 50 to 200 items. Similarities between the bin-packing and the stock cutting problems are depicted in [7].

4 Heuristics

As mentioned earlier, *Renault* proposes two different greedy heuristics (SG and LAG) for $RTLP$. In order to challenge their algorithms, local search methods are designed. In this context, comparisons is proposed between the *Renault* greedy algorithms and four tabu search algorithms.

4.1 Greedy heuristics

SG builds a solution from scratch, and at each iteration, select an item (following different possible rules) from a list L of non already inserted items, and adds it to the solution at minimum cost (i.e. which minimizes the augmentation of f, label 3 of Figure 1). This process stops when L is empty. In LAG, at each iteration, the algorithm tries each item j of L, and for each j, tries the next p (parameter) insertions following this possible insertion of j (look-ahead process). At the end of the iteration, the item j that would involve the lowest cost in the next p iterations is selected and inserted at the best position. As before, this process continues until L becomes empty. Both processes are fast (a few seconds per run), and therefore relevant to *Renault* as they are dealing with more than a thousand trucks to operate on a daily basis. As the truck loading plan has to be often rebuilt, due to constant changes in the production plan, the need for a fast and efficient algorithm is mandatory. The two above methods perform restarts as long as a time limit T is not reached. When T is reached, the best generated solution is returned.

4.2 Local search algorithms

In contrast with greedy algorithms, where a solution is built step by step, a local search method starts from an initial solution, and at each iteration, explores the neighborhood $N(s)$ of the current solution s to reach a possible better solution. The *neighborhood* of a solution s is the set of solutions reachable from s when performing a specific modification (called a *move*) on s. The neighborhood exploration continues until a stopping criterion is met (usually a number of iterations or a computing time limit), and the best found solution is returned to the user.

In this context, a *descent* algorithm performs the best move at each iteration, and stops when no improvement of the current solution is possible. In other words, a descent method stops when it reaches a local optimum. *Tabu search* (TS) was first proposed by Glover in [3] and is nowadays still considered as one of the most efficient method for exploring the search space. To prevent tabu search from being stuck in a local optimum, when a move is performed, TS forbids the reverse move for θ (parameter) iterations. To get a better understanding, a sketch of a generic tabu search is presented in Algorithm 1. A recent review on metaheuristics is proposed in [2], which is a good reference book in this area.

Algorithm 1 Generic tabu search

```
Generate an initial solution $s$ and set $s^\star = s$
While no stopping criterion is met, do
    1. Perform a move to reach the best non tabu neighbor solution $s' \in N(s)$
    2. Forbid the reverse move for $\theta$ iterations
    3. Set s = s'
    4. If $f(s) < f(s^\star)$, set $s^\star = s$
Return the best solution $s^\star$
```

4.2.1 Solution space

A decision has been made to use an *encoded* solution s while working with the local search methods. The encoded solution s is actually a list of elements. To build a solution and compute its quality, a *decoding greedy algorithm* (DGA) is performed on the encoded solution. It decodes the solution s into a real solution $\hat{s}$, and then returns the total length of the truck load f. To drive DGA, information (some are mandatory and others are optional) are carried in each element of s, and can contain the item identifier (ID, mandatory), the class identifier ($\mathcal{C}$, mandatory), the item orientation (O, optional), and the item side (S, optional). Thus, component i of the solution s takes the form $s_i = (ID_i, \mathcal{C}_i, O_i, S_i)$, where $ID \in \{1, \ldots, n\}$, $\mathcal{C} \in \{1, \ldots, m\}$, $O \in \{$*not rotated*, $90°$ *rotated*$\}$, and $S \in \{$*left-sided*, *right-sided*$\}$. DGA thus decodes the vector s into a real solution $\hat{s}$ by inserting in $\hat{s}$ the items from s in a $FIFO$ order, and using the O and S information (if provided) while respecting the class management constraints. At each step, DGA pops the next item i of s, greedily loads it in the truck, while respecting $\mathcal{C}_i$, O_i and S_i. If O_i (or S_i) is not provided,

DGA can decide by itself its value (minimizing the augmentation of f), and thus owns more freedom.

An example is now proposed. Five items, initially oriented as presented in Figure 2, have to be placed in a truck. A possible encoded solution is the following: *s=((1,1,not rotated,right-sided), (3,1,*$90°$ *rotated,right-sided), (2,1,*$90°$ *rotated,left-sided), (5,2,?,?), (4,2,?,right-sided))*. The corresponding decoded solution $\hat{s}$ is illustrated in Figure 3.

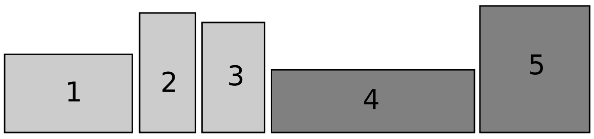

Figure 2: Items (with initial orientations)

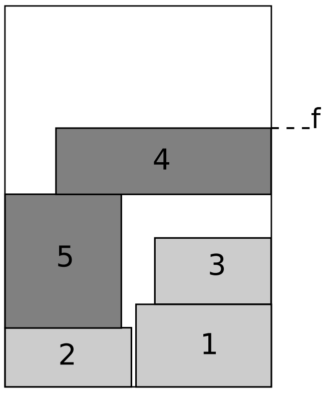

Figure 3: Decoded solution $\hat{s}$

To generate this solution, DGA performs the following steps: it pops the first element ID_1 and loads the corresponding item on the right side, without rotation. At that time, the next item, corresponding to ID_3, is loaded on the right with a $90°$ rotation. Then item ID_2 is loaded on the left with rotation whereas item ID_5 is inserted at the best possible position tried by DGA while respecting the class constraint. Finally item ID_4 is inserted on the right side but DGA decided its orientation. When DGA is over, it returns the value f, which is in this example the extremity of item ID_4

4.2.2 Neighborhood structures

To generate a neighbor solution s' from the current solution s, the seven following moves are possible:

- move an item j from position x to position y;
- move an item j from position x to position y while switching to the opposite value its orientation O;
- move an item j from position x to position y while switching to the opposite value its side S;
- move an item j from position x to position y while switching S and O to the opposite values;
- switch O to the opposite value;
- switch S to the opposite value;
- switch O and S to the opposite values.

All the moves are performed while respecting the class management constraint and ties are broken randomly.

4.2.3 Tabu search

Four different tabu search are proposed, denoted $TS1$, $TS2$, $TS3$ and $TS4$. The notation TS is simply used if it refers to common features of the four tabu search algorithms. TS starts from an initial encoded solution where items are ordered by decreasing areas.

$TS1$, $TS2$, $TS3$ and $TS4$ differ in the sense that for $TS1$, only the information ID and $\mathcal{C}$ are contained in each element of the encoded solution. $TS2$ contains ID, $\mathcal{C}$, and O. $TS3$ contains ID, $\mathcal{C}$, and S. $TS4$ contains ID, $\mathcal{C}$, O and S. Thus, for $TS1$, DGA can decide on its own the orientations and the sides (while focusing on the smallest augmentation of f). In $TS2$ and $TS3$, DGA has the order in which the insertion must be made, and the orientation or the side of each item (but not both). Finally $TS4$ constraints DGA at the maximum due to the complete information set contained in each component s_i of the vector s.

In tabu search, it is common that, at each iteration, only a fraction υ of the possible moves are generated. This restriction in the neighborhood size allows to perform more iterations for a same time limit, and helps in bringing more diversification. The neighborhood size υ is set to 50%, while, at each iteration, the tabu tenure θ is set to a uniformly distributed value between 25 and 55. Such values were determined by prior computational experiments.

5 Results

In this section are compared all the methods presented in this paper on a set of 30 real benchmark instances provided by *Renault*. Note that a total of

600 instances exist (with the same difficulty), and are available upon request to the authors. Tests were performed on an Intel Quad-core i7 @ 3.4 GHz with 8 GB DDR3 of RAM memory. The time limit T is set to 1800 seconds. Such a time limit was confirmed to be a relevant value, as the goal of this project is to benchmark the Renault algorithms by finding the best possible solutions. Thus, the computation time is not a hard constraint in this case. As tabu search is not a method with restarts, its results are averaged over five runs.

The results are summarized in Table 1. The first two columns indicate the values of n (number of items) and m (number of classes) of each instance. The third column ($f^\star$) corresponds to the objective function value of the best solution ever found by any of the algorithms. The following column reports the percentage gap between the solution of SG and $f^\star$. The next columns provide the same information for the other methods. For each instance, the best result is indicated in bold. The last rows give the average gap per method depending on m, in addition to the average times (in seconds, rounded to the closest integer) per method, and depending on m, to find the best solution. As we do not have time indications about SG and LAG, the corresponding cells are blank.

The superiority of LAG over SG is obvious, as the average gap decreases by a factor of more than five between the two methods. This is mainly due to the fact that LAG, with the parameter p, explores the impact of future possible insertions. On the opposite, SG does not have any indication on the short term consequences of the current insertion. TS does not show amazing improvements on the greedy heuristics. We conjecture that it is due to the fact that $f^\star$ is not far from the optimum.

$TS1$ is a method to avoid as it obtains poor results on most of the instances. This is probably due to the full freedom given to DGA to build a solution, when compared to the other algorithms. We can observe that $TS1$ is much better when the number of classes is one. In this case, the freedom given to DGA seems to be more relevant, as the class management constraint is not used and the number of tests to find a good insertion is more important.

$TS4$ is less competitive than $TS2$ and $TS3$. This can be easily explained by the fact that $TS4$ do not let DGA any choice on the solution building. It is thus not surprising that $TS4$ is the fastest method (regarding the time needed per iteration) of the TS family, as DGA does not have to perform a lot of test to insert an item at the best position. However, even if $TS4$ it is the fastest method per iteration, it is the slowest method to find its best solution.

Table 1: Computational results

n	m	$f^\star$	SG	LAG	$TS1$	$TS2$	$TS3$	$TS4$
23	1	12840	1.01	0.86	4.60	**0.06**	0.21	0.92
25	1	13000	3.85	0.31	3.85	**0.00**	0.05	0.98
24	1	12920	1.32	0.08	1.39	**0.00**	0.06	0.38
25	1	13040	3.76	0.31	3.76	**0.00**	**0.00**	0.60
26	1	13460	0.15	0.07	0.15	**0.00**	**0.00**	0.01
20	2	13610	6.83	1.25	14.11	**0.00**	1.25	0.53
23	1	12980	3.08	0.08	6.63	**0.00**	**0.00**	0.26
25	1	14120	4.02	4.01	3.75	0.07	**0.00**	0.26
18	4	13295	0.56	0.15	24.69	**0.00**	0.05	0.18
23	3	12852	5.23	0.50	18.67	**0.00**	0.03	0.16
20	2	13330	9.08	0.60	12.08	**0.00**	0.08	0.08
17	3	13070	0.23	0.15	14.77	**0.00**	**0.00**	**0.00**
25	1	13390	1.12	0.97	1.05	**0.00**	0.16	0.42
20	2	13150	7.53	1.52	12.09	**0.00**	0.61	0.67
20	4	13325	4.68	2.42	25.77	**0.00**	2.65	0.11
24	1	13010	3.69	0.46	3.61	**0.00**	**0.00**	0.52
23	4	12902	3.36	2.19	6.17	**0.02**	1.02	0.39
24	1	13380	1.12	0.60	4.11	**0.00**	**0.00**	0.04
24	1	13380	1.12	0.60	4.11	**0.00**	**0.00**	0.04
23	1	13040	3.68	0.23	6.75	**0.00**	0.05	0.26
25	1	13020	3.76	0.23	3.76	**0.00**	**0.00**	0.53
25	1	13380	0.82	0.60	0.75	**0.00**	**0.00**	0.20
24	1	13380	1.12	0.60	4.04	**0.00**	**0.00**	0.01
18	2	11530	10.41	0.95	11.88	**0.00**	**0.00**	**0.00**
23	1	12550	21.12	**0.00**	9.96	**0.00**	**0.00**	**0.00**
19	2	12170	6.98	0.33	9.20	**0.00**	**0.00**	**0.00**
23	1	13070	3.83	0.23	6.96	**0.00**	0.05	0.51
25	1	13380	0.75	0.60	0.75	**0.00**	**0.00**	0.12
20	1	13300	1.50	1.13	17.29	**0.00**	0.18	1.01
25	1	13080	3.59	0.08	3.59	**0.00**	**0.00**	0.28
AVG			**3.98**	**0.74**	**8.01**	**0.00**	**0.21**	**0.32**
AVG ($m = 1$)			**3.22**	**0.60**	**4.54**	**0.01**	**0.04**	**0.37**
AVG ($m > 1$)			**5.49**	**1.01**	**14.94**	**0.00**	**0.57**	**0.21**
AVG t					**176**	**232**	**240**	**645**
AVG t ($m = 1$)					**177**	**323**	**289**	**712**
AVG t ($m > 1$)					**173**	**48**	**141**	**513**

Results show that $TS2$ has the minimum gap on 29 instances over 30. Experiments clearly show that some freedom should be given to DGA, as $TS2$ and $TS3$ are the most powerful tabu search methods. Remember that the difference between $TS2$ and $TS3$ is the carried information O or S. Results show that O is the most important feature to carry in the encoded solution s. This can be explained in the sense that the orientation O has an important impact, as it clearly drives $\hat{s}$ by forcing the orientation of each item.
The average times show that if $m > 1$, $TS2$ converges very quickly to a promising local optimum. For such instances, the performance of $TS3$ decreases as it is even outperformed by $TS4$. This is probably due to the fact that there is less possibility to move the items in the solution, as the class management constraint must be respected, and thus allows $TS2$ to find the local optimum faster.

Figure 4 shows a plot indicating for each average time x (horizontal axis), the total number y (vertical axis) of trucks (among 30) requiring less than x seconds to find its best solution. We can observe that fifteen instances require less than 100 seconds. Note that the average time over all the trucks is 175 seconds, and that the maximum time for a single truck was found to be 663 seconds. As mentioned above, the time limit T is set to 1800 seconds for all the algorithms, thus there exists a lot of spare time in the $TS2$ algorithm (and in the other TS methods as well). A diversification procedure could be a relevant feature to improve the search and better use the remaining time.

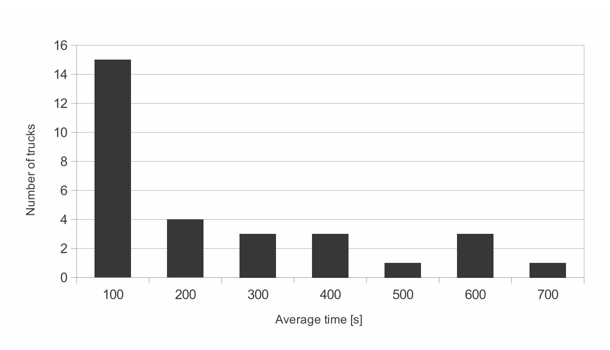

Figure 4: Average time needed by $TS2$ to provide its best solutions based on a time limit of 1800 seconds

6 Conclusion

In this paper, we have proposed a practical application daily faced by the French car manufacturer *Renault*. We compared their algorithms to more sophisticated methods and showed that tabu search is powerful. Future work includes adding diversification and intensification procedures to tabu search to improve its performance. In addition, genetic algorithms have proven their capabilities to give good results on the bin-packing problems, a hybrid genetic algorithm should be proposed in conjunction to tabu search. Moreover, as we already have different neighborhoods, a variable neighborhood search (VNS) could as well be easily set up and tested.

Acknowledgments

The authors would like to thank Alain Nguyen and Jean-Philippe Brenaut, both from the *Renault* operations research team, for their availability and relevant advices on the elaboration of the algorithms.

References

[1] T. G. Crainic, G. Perboli, and R. Tadei. TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009.

[2] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer, 2010.

[3] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1:190–205, 1989.

[4] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128(1):34–57, 2001.

[5] A. Khanafer, F. Clautiaux, and E.-G. Talbi. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & Industrial Engineering*, 39(1):54–63, 2012.

[6] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. Exhaustive approaches to 2D rectangular perfect packings. *Information Processing Letters*, 90(1):7–14, 2004.

[7] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[8] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166, 1999.

[9] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, April 1999.

[10] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15:310–319, 2003.

[11] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[12] A. Nguyen and J-Ph. Brenaut. A truck loading algorithm for Renault's supply chain system. In *23rd EURO Conference, Bonn, Germany*, July 5-8, 2009.

[13] N. Ntene and J.H. van Vuuren. A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem. *Discrete Optimization*, 6(2):174–188, 2009.

[14] D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167, 2005.

[15] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3):1304–1327, 2007.

[16] M.C. Riff, X. Bonnaire, and B. Neveu. A revision of recent approaches for two-dimensional strip-packing problems. *Engineering Applications of Artificial Intelligence*, 22(45):823–827, 2009.